BPSWeb
Projeto de escalonamento de horários por turnos

Susana Filipa Neves Pedro João

Mestrado Integrado em Engenharia de Redes e Sistemas Informáticos

Departamento de Ciência de Computadores

2013

Orientador

Engº Armando Barbosa, Bullet Solutions - Sistemas de Informação, SA

Coorientador

Prof. Sandra Alves, Departamento de Ciência de Computadores,
Faculdade de Ciências da Universidade do Porto

U.PORTO

FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Todas as correções determinadas pelo júri, e só essas, foram efetuadas.

O Presidente do Júri,

Porto, _____/_____/_____

Agradecimentos

Agradeço a todos os que contribuíram para a ideia e para o desenvolvimento deste projeto.

A toda a equipa da Bullet Solutions agradeço a boa receção e o bom ambiente ao longo de todo o estágio. Em particular ao Eng. Armando Barbosa e ao Eng. Luís Costa agradeço a disponibilidade na orientação e no esclarecimento de dúvidas no desenvolvimento do projeto, bem como ao Rui Fernandes pelo auxílio na elaboração do *design* da interface.

Um agradecimento também à professora Sandra Alves pela disponibilidade de orientação na escrita deste relatório.

Por fim, agradeço à minha família e amigos mais próximos o apoio incondicional.

Resumo

O presente relatório de estágio descreve o trabalho realizado durante sete meses numa empresa de desenvolvimento de soluções informáticas de otimização, a Bullet Solutions. O trabalho consistiu no desenvolvimento de um projeto *web* de escalonamento de horários por turnos denominado *Bullet Personal Scheduler Web* (BPSWeb). Esta aplicação, distribuída na forma de *Software* como um Serviço, é um novo produto da empresa para resolução automática e otimizada de problemas de alocação de pessoal a um turno num dia e num local previamente definidos.

Para alcançar os objetivos foi analisado o mercado concorrente, desenhada a arquitetura da solução, definida uma base de dados sólida capaz de suportar todo o projeto e implementada a solução.

O resultado foi uma aplicação *web* capaz de tratar todos os dados importantes e gerar uma solução de alocação de recursos a necessidades de uma instituição.

Abstract

This report describes the work of a seven month internship at Bullet Solutions, a company that develops optimization software solutions. The internship consisted in the development of the *Bullet Personal Scheduler Web* (BPSWeb), a shift scheduling web application project. This application, distributed as Software as a Service, is a new company's product that solves automatically and efficiently problems of staff allocation to a predetermined day and place.

To achieve these goals the rival market solutions were analyzed, followed by the design of the architecture and the design of the core databases that support all project, and finally the solution was implemented.

The final result was a web application capable of dealing with all the important data and capable of generating a resource allocation solution that meets the needs of a specific institution.

Conteúdo

Agradecimentos **3**

Resumo **4**

Abstract **5**

Lista de Tabelas **8**

Lista de Figuras **10**

1 Introdução **11**
1.1 Enquadramento da Empresa 12
1.2 Objetivos 12
1.3 Estrutura do relatório 13

2 Preliminares **15**
2.1 Escalas de trabalho por turnos 15
2.2 Estado de Arte: *Software* existente 18
2.3 Conceitos Auxiliares Relevantes 19
2.3.1 *Software* como um Serviço (SaaS) 19
2.3.2 Arquitetura Orientada a Serviço 20
2.3.3 Serviço Web 21

3 Arquitetura da Solução **22**
3.1 Casos de Uso 24
3.2 Bases de Dados 25

4 Implementação **33**
4.1 Tecnologia 33
4.2 Desenvolvimento 35
4.2.1 Base de Dados 36
4.2.2 Formulários *Scheduler* 39

4.2.3 Formulários *FrontEnd* 41
4.2.4 Formulários JQuery 44
4.2.5 Preparação para multi linguagem 45

5 Resultados **46**
5.1 Formulários da componente *Scheduler* 46
5.2 Formulários básicos da componente *FrontEnd* 47
5.3 Formulários secundários da componente *FrontEnd* 48
5.4 Comparação com *Softwares* Existentes 51

6 Conclusão **52**
6.1 Trabalho Futuro 53

7 Acrónimos **54**

Referências **55**

Lista de Tabelas

4.1 Funções no Repositório para quando o utilizador pretende editar um local de trabalho. 42
4.2 Excerto do código em Razor HTML para mostrar a edição de um local de trabalho 43

Lista de Figuras

2.1 Modelo de trabalho *6 on 6 off*; 17
2.2 Modelo de trabalho Continental; 17

3.1 Modelo de arquitetura segundo as Tecnologias de Informação 22
3.2 Desenho da arquitetura segundo o Modelo das Três Camadas 23
3.3 Diagrama dos casos de uso. 24
3.4 Diagrama de Atividade relativo aos casos de uso. 25
3.5 Diagrama ER da Base de Dados BPSWeb 26
3.6 Parte do Modelo ER da Base de Dados de um cliente (Configurações Iniciais). 27
3.7 Parte do Modelo ER da Base de Dados de um cliente (Configurações Secundárias) 28
3.8 Parte do Modelo ER da Base de Dados de um cliente (tabelas geradas pelo algoritmo). 29
3.9 Relacionamento da tabela Periodo com as restantes tabelas. 29
3.10 Parte do Modelo ER da Base de Dados de um cliente. 30
3.11 Tabelas *WeekDay* e *WeekDay_Locale* e respetivos registos 31

4.1 Estrutura do Modelo MVC 34
4.2 Exemplo de duas tabelas criadas no SQL Server 37
4.3 Instrução SQL com uso da chave primária (lado esquerdo) e do índice (lado direito). 38
4.4 Plano de execução para a instrução com uso da chave primária (lado esquerdo) e do índice (lado direito). 39
4.5 Comportamento da classe *BusinessLogic* no *Scheduler* 40
4.6 Interface gráfica do formulário das Disponibilidades 44

5.1 Interface gráfica do projeto *Scheduler*. 46
5.2 Interface gráfica do formulário de inserção de um novo período. 47
5.3 Interface gráfica da listagem dos Locais de Trabalho 47
5.4 Interface gráfica da matriz de custo turno/categoria profissional. 48
5.5 Interface gráfica dos detalhes das Necessidades. 48

5.6 Interface gráfica de uma geração em curso. 49
5.7 Interface gráfica da Agenda filtrada por um local de trabalho. 50
5.8 Interface gráfica da Resolução da Solução Incompleta. 50

Capítulo 1

Introdução

Um sistema de informação seja ele de geração ou tratamento de dados é, nos dias de hoje, indispensável. O mundo tecnológico está diversificado em todas as áreas com que lidamos direta ou indiretamente e a área do trabalho não é exceção. Com a evolução do trabalho como exercício de uma profissão, a organização de uma instituição e do horário dos seus trabalhadores é fundamental. Com isso surgem as escalas de horários que tanto verificamos em inúmeras profissões, tais como na área da saúde, nos transportes, nos restaurantes, nas escolas ou na polícia.

Muitas vezes, devido às enormes restrições e necessidades que uma organização possui, torna-se muito difícil elaborar uma escala de trabalho sem a ajuda de um sistema informático. O número de colaboradores, as suas disponibilidades, necessidades de reforços em determinada situação, novas regras ou políticas impostas que não estavam previstas, são pequenos exemplos que condicionam a solução e consequentemente obrigam muitas vezes a uma solução que não era a desejada.

Neste contexto, surge a ideia do desenvolvimento de uma aplicação que resolva este problema, através do recurso a sistemas de apoio à decisão, ou seja, através de um algoritmo que calcule os pesos das alternativas possíveis e tome uma decisão quanto à melhor dessas alternativas a fim de alcançar o melhor resultado possível.

Um sistema intuitivo, que possa fazer a geração automática e otimizada do escalonamento de horários por turnos, poderá poupar tempo e consequentemente trazer melhores condições de trabalho para todos os que são inseridos nesta área. Com uma aplicação deste nível, uma instituição poderá fazer a gestão de todos os colaboradores e equipas, categorias profissionais e qualificações, definir as disponibilidades de cada um, consultar o banco de horas, definir as férias e ausências a curto e longo prazo, criar regras e especificar as necessidades que a empresa precisa para o bom funcionamento. O programa também será capaz de gerar uma solução para o conjunto de objetivos e restrições

especificadas, obtendo desta forma um horário final, sem nunca esquecer a segurança e privacidade importantes em qualquer serviço desta área.

Este relatório descreve o trabalho ao longo do estágio curricular para o desenvolvimento de uma solução com estas características.

1.1 Enquadramento da Empresa

A Bullet Solutions é uma empresa com sede no Porto que se dedica ao desenvolvimento de soluções informáticas de otimização para problemas combinatórios à medida [1]. Nascida em 2006, o primeiro *software* desenvolvido baseia-se no escalonamento de horários de instituições universitárias, o Bullet TimeTabler Education (BTTE). Com base em informações facultadas pela instituição (como as salas, os professores e as disciplinas) e através de um conjunto de restrições necessárias ao bom funcionamento da mesma (como por exemplo, a disponibilidade de horários de cada professor ou a necessidade da aula de uma disciplina só poder ser dada em determinada sala) existe um algoritmo que constrói uma solução para atribuir professores, turmas e salas a um possível horário. Além do BTTE, exemplos de outras soluções da Bullet Solutions são o Bullet TimeTabler Medical (responsável pela geração e gestão de escalas médicas), o Bullet Callendar (complementar ao BTTE e que permite gerir diariamente eventos pontuais de uma instituição) e o Bullet Scheduler (apoia os responsáveis de produção nas tarefas de sequenciamento, controlo e adaptação a mudanças do chão da fábrica).

1.2 Objetivos

Para conseguir alargar a oferta de soluções e possibilitar a geração de horários de profissões que trabalham por turnos, surgiu a ideia de ser desenvolvido o Bullet Personal Scheduler Web (BPSWeb). Este projeto baseia-se no mesmo conceito dos anteriores, possibilitando um vasto conjunto de personalização de informações. Assim, o objetivo é a criação de um projeto *web* de escalonamento automático de horários por turnos, preparado para múltiplos clientes e para múltiplas línguas.

O trabalho desenvolvido neste estágio curricular consiste na implementação de três partes do projeto: a Base de Dados, uma ferramenta interna de gestão de clientes e servidores e a interface gráfica de auxílio ao cliente. As funcionalidades que o BPSWeb no fim deverá possuir são as seguintes:

- Gestão dos colaboradores, suas qualificações e categorias profissionais;
- Gestão dos turnos;
- Gestão de ausências;
- Gestão de bolsas de horas;
- Gestão de utilizadores;
- Definição de uma matriz de custo turno/categoria profissional;
- Definição da disponibilidade de cada colaborador;
- Marcação de necessidades;
- Validação de dados;
- Notificação para os utilizadores;
- Listagem de relatórios e horários;
- Gestão das alocações de colaboradores a necessidades;
- Resolução da solução incompleta.

De forma a atingir os objetivos enumerados, foram definidas as seguintes fases de desenvolvimento do projeto:

1. Estudo e análise do mercado concorrente;
2. Definição das funcionalidades;
3. Desenho da arquitetura do sistema;
4. Desenho e implementação das base de dados;
5. Primeira versão do *FrontEnd*;
6. Ajustes e melhorias.

1.3 Estrutura do relatório

Este relatório além da presente introdução (capítulo 1), apresenta na sua estrutura mais cinco capítulos organizados da seguinte forma: no capítulo 2 são explicados os conceitos do sistema de trabalho por turnos, são apresentados os *softwares* concorrentes desta área no mercado atual e ainda o conceito de *Software como um Serviço*, uma vez que é um importante modelo de negócio que será usado para a comercialização do produto final.

O capitulo 3 faz referência à arquitetura do projeto e dessa forma como vai funcionar toda a aplicação e como se relacionará com a base de dados.

No capítulo 4 são explicados todos os aspetos relativos à implementação, desde as tecnologias e ferramentas utilizadas à implementação do sistema.

No capítulo 5 são apresentados os resultados finais com as capturas dos principais ecrãs e uma pequena comparação do trabalho final com os *softwares* concorrentes. Por fim, no capítulo 6 tiramos conclusões e fazemos uma análise do possível trabalho futuro.

Capítulo 2

Preliminares

Neste capítulo vamos descrever o tema e o problema do projeto bem como as soluções disponíveis no mercado para esses problemas. Também serão descritos conceitos relacionados relevantes para o desenvolvimento do BPSWeb.

2.1 Escalas de trabalho por turnos

O trabalho, como exercício de uma atividade profissional, é um conceito histórico, cuja definição tem sofrido alterações ao longo dos tempos. A forma como um conjunto de indivíduos se organizam para gerar um produto difere de época para época, mas com a necessidade de garantir segurança durante um período da noite ou de produzir durante 24 horas para satisfazer necessidades, surge o trabalho por turnos. Também conhecido como *Shift Work*, este tipo de trabalho consiste numa prática de emprego em que o dia é dividido por turnos (*shifts*) para fazer uso de todas as horas do dia e para cada um desses turnos é definido um conjunto de trabalhadores. Algumas das profissões que trabalham por turnos estão relacionadas com serviços ao público, como é o caso de médicos e enfermeiros, polícia, serviço de transportes ou mesmo restauração [2].

Para que os trabalhadores tenham condições de trabalho, são exigidos inúmeros formalismos às entidades, tais como: a duração de trabalho não pode exceder o limite máximo dos períodos normais de trabalho; são obrigatórios descansos semanais; só depois do descanso semanal é que o trabalhador pode trocar de turno; entre outros. Estas exigências ainda tornam mais complicado o desenvolvimento e a gestão de um horário, uma vez que pode haver um conjunto de restrições para cada trabalhador.

Com o objetivo de poupar tempo e evitar erros humanos, têm vindo a ser desenvolvidos sistemas de apoio à decisão para ajudar o utilizador a adotar decisões inteligentes. A maioria dos *softwares* existentes no mercado simplifica a gestão dos sistemas de escalas de trabalho, através da automatização de todo o processo, tornando o planeamento de horários mais simples e fácil, independentemente da complexidade dos turnos. Para que um sistema deste tipo cumpra os objetivos, deve ter funções básicas como: armazenar informações sobre todos os empregados, possibilitar a escolha dos turnos para cada empregado e respeitar as requisições feitas (por exemplo, não permitir que um empregado trabalhe mais do que um número específico de horas seguidas, ou definir o tempo mínimo que deve ter de almoço). Assim, podemos dizer que um bom sistema de escalonamento por turnos deve possuir cinco caraterísticas principais: duração e tipo de turno, modelo de trabalho *on-off*, horas extras e políticas de horários. Além destes importantes aspetos, é valorizado um sistema que seja intuitivo, de fácil utilização e de pouco consumo de recursos [3] .

De forma a melhor compreender a implementação que se pretende, foram estudados os principais tipos de turno que existem, que podem ser distribuídos da seguinte forma [4]:

- **Fixo ou permanente** (*fixed/permanent*): cada pessoa trabalha todos os dias no mesmo horário, por exemplo, só durante o dia ou só no turno da noite;
- **Rotativo** (*rotating*): cada pessoa trabalha em vários turnos. A rotação pode ser Lenta (maior que semanalmente e geralmente trabalhando em torno de 21 dias no mesmo turno) ou Semanal (entre 5 a 7 dias para cada turno).
- **Oscilante** (*oscillating*): o trabalhador alterna entre turnos da noite e do dia ou então entre tarde e noite em base semanal;
- **Turno Interrompido** (*split shift*): uma pausa de algumas horas separa as horas de trabalho feitas no mesmo dia, por exemplo, os trabalhadores na gastronomia ou no setor de transportes, onde há picos maiores de movimento em certos horários;
- **Turnos Substitutos** (*relief shifts*): a pessoa pode entrar em qualquer um dos padrões acima, mas o horário depende do horário do trabalhador que faltou;
- **Tipos Alternativos**: semana de trabalho de 4 dias ou períodos de trabalho de 12 horas. Podem ser usados em operação de 1 turno, 2 turnos ou 3 turnos, contínua ou descontínua, isto é, com respeito aos fins-de-semana.

Fatores como o tipo de turno, a cultura de cada país, o número de horas de descanso necessárias ao trabalho bem como as necessidades de cada profissão, influenciam o horário que se pode criar. Existem diversos padrões de turnos. São descritos agora os

mais comuns.

Para turnos de 12h:

- Modelo de 4 dias: 12 horas de trabalho seguidas de 24 horas de descanso, 12 horas de trabalho e outras 48 horas de descanso.
- Modelo de 6 dias: 12 horas de trabalho seguidas de 12 horas de descanso nos primeiros quatro dias. O quinto e o sexto dia são folgas (48 horas sem trabalhar).
- Modelo de *6 on, 6 off*: nos primeiros três dias o trabalho é na primeira metade do dia; nos três dias seguintes, o trabalho é na segunda parte do dia; nos seis dias seguintes descansa-se, como mostra a Figura 2.1. A, B, C e D representam empregados.

Turno	Semana 1							Semana 2				
	Seg	Ter	Qua	Qui	Sex	Sab	Dom	Seg	Ter	Qua	Qui	…
06:00 - 18:00	A	A	A	B	B	B	C	C	C	D	D	…
18:00 - 06:00	D	D	D	A	A	A	B	B	B	C	C	…

Figura 2.1: Modelo de trabalho *6 on 6 off*;

Para turnos de 8h:

- Modelo de 5 dias: ao fim de 4 dias seguidos a trabalhar, o quinto corresponde a uma folga. Para que os empregados não folguem todos no mesmo dia, para cada empregado a primeira folga começa num dia de semana diferente.
- Modelo Continental: trabalha os três primeiros dias no turno da manhã (6h-14h), dois dias no turno da tarde (14h-22h) e dois dias à noite (22h-6h): no fim dos 7 dias, tem de ter uma folga, caso contrário, faz mais do que 8h seguidas.
- Dois dias, duas noites, quatro folgas. Como o próprio nome indica, após quatro dias a trabalhar na primeira metade ou na segunda metade do dia, descansa quatro dias.

Turno	Seg	Ter	Qua	Qui	Sex	Sáb	Dom
06:00 - 14:00	A	A	A	B	B	C	C
14:00 - 22:00	C	C	C	A	A	B	B
22:00 - 06:00	B	B	B	C	C	A	A

Figura 2.2: Modelo de trabalho Continental;

Para turnos de 24h: o modelo pode ser T-F-T-F-F-T-F ou T-F-F-F-T-F-F-F, onde T corresponde ao período de Trabalho e F ao período de Folga.

2.2 Estado de Arte: *Software* existente

Com o objetivo de conhecer o mercado de trabalho que propõe soluções para o problema da gestão de horários por turnos, foram analisados alguns *softwares* existentes. Aqueles que possuem uma versão temporariamente gratuita foram instalados e analisados. Conclui-se rapidamente que todas estas aplicações fazem apenas a gestão manual, não havendo qualquer algoritmo de geração ou otimização dos horários.

- ***Snap Schedule*** [5]: faz o planeamento e gestão de horários de trabalho para turnos de 8, 10 ou 12 horas, fixos ou em escalas. Também suporta horários rotativos. Permite importação de ficheiros *Comma Separated Values (.csv)*. Têm imensas funcionalidades, tais como a criação de diversos utilizadores, separação das permissões de administrador, marcação de férias e gestão de salários consoante o turno. No entanto, devido ao excesso de funcionalidade e falta de organização das mesmas, torna-se bastante confuso o seu funcionamento.

- ***Shift Planning*** [6]: uma aplicação *online* com capacidade de até 99 empregados que possui também aplicação móvel. De todos os *softwares* testados foi o que apresentou mais vantagens. Os seus pontos fortes relacionam-se com o facto de ser extremamente intuitivo e de permitir avisar o utilizador quando o horário escolhido não corresponde às suas preferências. O maior ponto fraco é não ter qualquer gerador automático de horário.

- ***Time Forge*** [7]: também é uma aplicação que corre no *browser*. Tem mais de 40 opções de relatório. Permite mensagem de texto e correio electrónico, bem como publicação *online*. Peca pela falta de uma boa estrutura gráfica e pela falta de gerador automático dos horários para as restrições de cada utilizador.

- ***Nimble Schedule*** [8]: muito fácil de usar e disponível em dispositivos móveis, esta também é uma ferramenta *online*. Pode ser sincronizada com o Facebook, Microsoft Outlook e Google Calendar.

Ainda foram testados outros produtos cujas vantagens sobre estes se mostraram irrelevantes. Confusos e sem escalonamento automático, alguns exemplos são: *Schedule it* [9], *Whentowork* [10] e *Timecenter* [11]. Um gestor de horários português que foi encontrado, tem como nome SISCOG: *"Dedica-se ao desenvolvimento de soluções informáticas para planear e gerir escalas de pessoal em empresas e organizações que oferecem serviços ao público em horário alargado"* [12]. Não foi testado por falta de versão gratuita, mas tudo indica que é um bom *software*, com um escalonamento de turnos otimizado, que faz a gestão não só de horários, mas de toda a organização de várias empresas do mercado de transportes.

2.3 Conceitos Auxiliares Relevantes

De seguida vamos apresentar algumas noções de modelo de negócio relevantes para este projeto.

2.3.1 *Software* como um Serviço (SaaS)

SaaS é uma forma de distribuir e comercializar *software* [13]. O fornecedor é responsável pela estrutura necessária e disponibilização do *software* e o cliente utiliza-o através do acesso à Internet. Não paga pela aquisição do produto, mas sim pela utilização do serviço ou de algumas das suas funcionalidades. É um modelo muito comum de negócio que pode incluir contabilidade, gestão de sistemas de informação, gestão de recursos humanos, entre outros. É ainda um conceito que fornece uma API para acesso pela *web*, através de Serviços de *Web*, interfaces REST (*Representational State Transfer*), SOAP (*Simple Object Access Protocol*) e outros protocolos. A maior vantagem é diminuir os custos da empresa através de manutenção por terceiros (*outsourcing*) de *hardware* e *software* e suporte de fornecedor SaaS. Outros proveitos desta solução são:

- Suporte técnico mais rápido, uma vez que não há necessidade de deslocação de uma equipa;
- *Multi-tenant*, ou seja, uso da estrutura por diversos clientes e empresas de forma simultânea;
- O procedimento é muito simples: o cliente apenas precisa de um identificador/palavra-passe para a aplicação. Isto reduz o tempo de execução do projeto bem como as despesas;
- Acesso universal, a partir de qualquer lugar com Internet.

Em contrapartida, os inconvenientes são:

- Fraca confiança nos Fornecedores de Serviços na Nuvem e dúvidas sobre a segurança de dados e confidencialidade de informações;
- Uma vez que as aplicações estão na nuvem, longe das aplicações dos utilizadores, pode haver latência no ambiente de desenvolvimento;

Uma solução SaaS é indicada principalmente para pequenas e médias empresas, uma vez que é uma boa solução tecnológica sem necessidade de investimento em *hardware* e infraestrutura. Quando desenvolvido de forma eficiente, possibilita ganhos de receita mais viáveis do que a solução comum a longo prazo. No início do seu desenvolvimento é importante pensar na infraestrutura, onde e como vai ser hospedado o *software*, como

fornecer o suporte aos utilizadores, qual a taxa que vão pagar e como ter a certeza que vai funcionar em todos os *browsers*. Também é necessário ter alguns cuidados, tais como: o *software* deve ser flexível de forma a permitir a sua configuração; no desenvolvimento, o processo de seleção e descoberta dos serviços deve ser intuitivo; deve permitir a alteração de um serviço enquanto é executado.

Deve ter-se em atenção que podem haver vários clientes para o mesmo projeto, mas cada cliente deve sentir-se tão confortável como se fosse o único cliente desse mesmo projeto. Assim, a configuração e personalização são questões cruciais no desenvolvimento de um SaaS [14]. Estes são dois conceitos diferentes. Enquanto "configuração" utiliza parâmetros pré-definidos para alterar as funcionalidades do *Software*, "personalização" requer alterações no código fonte.

Exemplos de boas aplicações desenvolvidas com o conceito de SaaS são as aplicações da Google (correio eletrónico, calendário, documentos) e a plataforma Amazon Web Service.

2.3.2 Arquitetura Orientada a Serviço

Para entender melhor SaaS será necessário compreender outros conceitos associados, tais como Arquitetura Orientada a Serviço (SOA) e Serviço Web [15]. O conceito de SOA é um modelo de arquitetura em que a solução lógica é apresentada como serviço. É desenvolvido para ser altamente intuitivo, ágil e produtivo. Fornece um suporte para tirar partido das vantagens dos princípios de orientação a serviços. De uma forma simplificada podemos definir como uma arquitetura projetada para satisfazer as necessidades de negócios de uma organização.

SOA não é uma tecnologia, mas sim uma abordagem de arquitetura para criar sistemas construídos a partir de serviços autónomos. É uma solução de design independente de qualquer produto, tecnologia ou industria. Diversas tecnologias, produtos e aplicações usam a implementação SOA, pois torna os serviços mais flexíveis, a manutenção mais fácil e o desenvolvimento mais otimizado. A arquitetura orientada a serviços também se relaciona com um conjunto de políticas e boas práticas para criar um processo para facilitar a tarefa de encontrar, definir e gerir os serviços disponibilizados.

Então qual é a diferença entre SaaS e SOA? Enquanto SaaS é um modelo de negócio, SOA é uma estratégia de arquitetura. Com um SaaS, em vez do cliente instalar a aplicação na sua máquina, esta é hospedada num servidor e o cliente acede a essa aplicação através da Internet pelo pagamento de uma taxa ao fornecedor. SOA é algo semelhante mas em pequena escala. Não fornece um modelo de negócio, mas sim pequenos processos isolados como serviços. É um estilo de arquitetura para construção de *software*. A ideia é construir

uma aplicação, juntando um conjunto de serviços em rede sem estado e reutilizáveis (como por exemplo, através de um Serviço Web).

Assim, podemos dizer que SOA fornece um serviço a outras aplicações, enquanto SaaS fornece um serviço a utilizadores. Podemos desta forma, juntar os dois conceitos para o desenvolvimento do projeto que queremos. Com SaaS fornecemos serviços a mais clientes. Construindo o projeto sob uma arquitetura SOA torna a aplicação mais fácil de escalar.

2.3.3 Serviço Web

Outro conceito importante a compreender é o de Serviços Web, que consiste numa solução utilizada para integrar serviços e fazer comunicação entre diferentes aplicações através da Internet [15]. Não é o mesmo que SOA, é por sua vez uma forma de implementar SOA. Isto é, um sistema que utilize SOA pode disponibilizar os seus serviços através de Serviços Web.

Estes componentes possibilitam que as aplicações enviem e recebam dados em formato XML, ou seja, qualquer que seja a linguagem que a aplicação tenha é traduzida para uma linguagem universal (XML). Desta forma, é possível uma aplicação invocar outra para executar tarefas mesmo que as duas aplicações estejam em diferentes sistemas e linguagens diferentes.

Serviços Web são utilizados muitas vezes para integração de aplicações de uma empresa, como por exemplo, correio eletrónico com os clientes e fornecedores, tendo ainda a possibilidade de otimizar e controlar os processos e tarefas de uma organização. Podemos assim dizer que Serviços Web permitem a organização de diferentes aplicações, fornecedores e plataformas.

Capítulo 3

Arquitetura da Solução

Neste capítulo é descrita a arquitetura desenhada para o projeto. A Figura 3.1 é relativa às Tecnologias de Informação e diz respeito à gestão de processos executados sempre que é feito um pedido pelo cliente. Como podemos observar, do lado do cliente existe o Navegador Web e do lado do servidor vamos ter um Servidor de Aplicações (*Application Server*), um Servidor de Base de Dados e vários *Workers*. O Servidor de Aplicações será responsável por receber os pedidos feitos pelo cliente e reencaminhar esses pedidos para o Servidor da Base de Dados. Este, por sua vez vai fazer toda a gestão dos processos, como verificação da disponibilidade dos *Workers* para executar o processo. Por fim, os *Workers* são servidores virtuais que executam todos os cálculos que o cliente pretende efetuar.

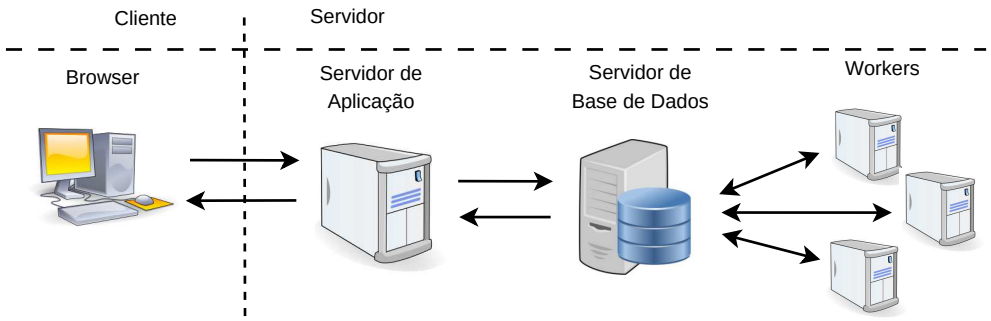


Figura 3.1: Modelo de arquitetura segundo as Tecnologias de Informação

Para fazer a gestão dos pedidos feitos pelos clientes, existe uma fila onde são guardados esses pedidos, uma lista de prioridades e um conjunto de estados com informação sobre a fase de execução em que o pedido se encontra.

Para melhor perceber o funcionamento desta arquitetura, vamos seguir o exemplo de quando um cliente deseja saber o resultado de uma operação algébrica. O cliente começa

por se autenticar. O Servidor de Aplicações recebe o pedido de autenticação do cliente e reencaminha esse pedido para o servidor de base de dados. Este servidor é responsável pela validação das credenciais do cliente, por isso faz uma consulta à base de dados e confirma ou rejeita as mesmas. Depois de autorizado, o cliente procede e faz o pedido para saber o resultado da operação. O Servidor de Aplicações recebe o pedido e faz novamente a ligação ao Servidor de Base de Dados. Neste segundo servidor, o processo é posto em fila de espera, é-lhe atribuída a prioridade que o cliente tem e inicializado o estado como ”na fila”. Assim que se verificar que um *Worker* está disponível é atribuído o respetivo *Worker* ao processo e altera-se o estado do processo para “a calcular”. Depois de terminado o cálculo pretendido o estado passa para o “calculado” e posteriormente é enviada a informação para o primeiro servidor e depois para o Navegador do Cliente.

A Figura 3.2 representa a arquitetura da solução segundo o Modelo das Três Camadas. Como podemos observar e como o próprio nome indica, temos três elementos. A interface do utilizador, também conhecida como UI, é onde o cliente faz as requisições das ações que pretende (é o espaço de interação entre a pessoa e as funcionalidades do programa). A camada de negócio ou lógica empresarial é a camada onde se definem as funções e regras de negócio. A camada de dados é a ponte entre a camada de negócio e a base de dados e por isso contem a definição das chamadas à BD.

Com esta estrutura de camadas independentes umas das outras, tornam-se mais simples possíveis alterações ou atualizações futuras que tenham de ser realizadas. No caso de alterações à BD, são apenas necessárias alterações na camada de dados, mantendo-se as restantes camadas indiferentes a esta alteração. Por exemplo, se uma instrução SQL, por algum motivo tiver de ser alterada, apenas será necessário proceder a essa alteração na camada de dados, poupando tempo e trabalho aos programadores. Também será mais fácil o diagnóstico de erros e a realização de testes.

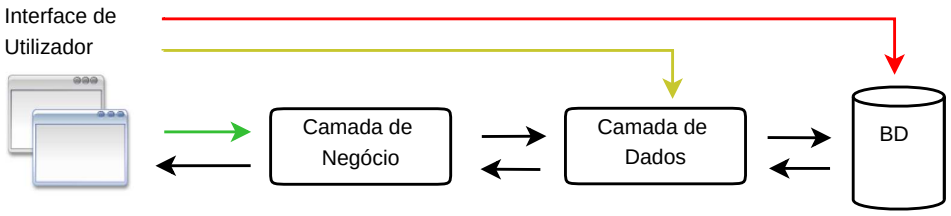


Figura 3.2: Desenho da arquitetura segundo o Modelo das Três Camadas

A razão porque as setas do diagrama são representadas com cores distintas nas relações entre a Interface do Utilizador e os restantes elementos relaciona-se com os cuidados a ter na implementação. Isto é, quando é implementada uma função para que o cliente possa

executar uma ação, essa função deve levar o processo a consultar a camada de negócio, depois a camada de dados e por fim o acesso à BD. Se se ignorar uma das camadas é quebrada a lógica da arquitetura e não é cumprida a responsabilidade de cada camada.

Como este é um projeto bastante complexo, na medida que tem como objetivo abranger um grande número de diferentes indústrias, é fundamental que o início do seu desenvolvimento seja seguro e flexível. Com grande probabilidade terão se ser feitas diversas alterações consoante as exigências de cada cliente. Por esta razão, existem diversas metodologias que foram adotadas que fornecem a flexibilidade e segurança pretendida.

3.1 Casos de Uso

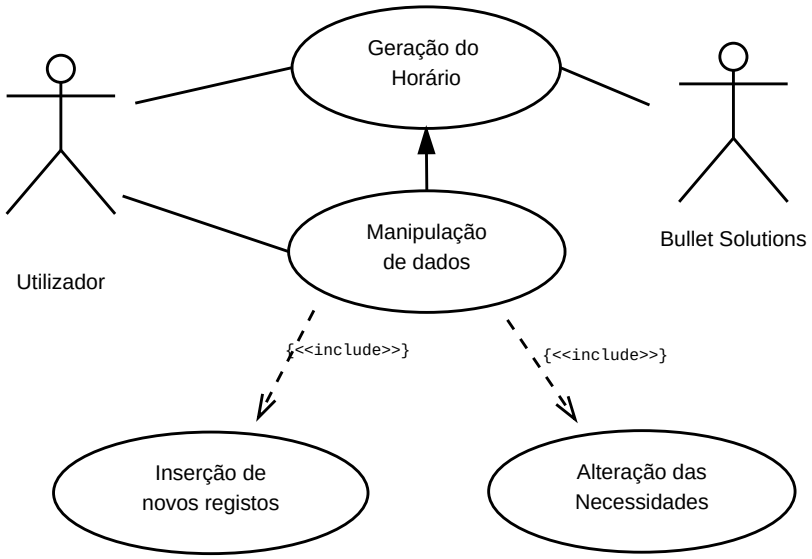


Figura 3.3: Diagrama dos casos de uso.

A figura 3.3 representa o diagrama dos casos de uso do projeto, através do qual podemos identificar os atores - utilizador cliente e um servidor da empresa - e os requisitos funcionais do sistema - manipulação de dados e geração de horários. Sobre estes requisitos, podemos analisar mais especificamente as atividades na figura 3.4.

Depois do utilizador estar autenticado vai definir o período para o qual pretende obter um horário, que pode ser um mês, um semestre, um ano ou qualquer outro intervalo de tempo desejado. Seguidamente, para esse período, vai inserir (ou editar ou ainda consultar ou remover) registos da configuração da sua empresa: locais de trabalho, qualificações,

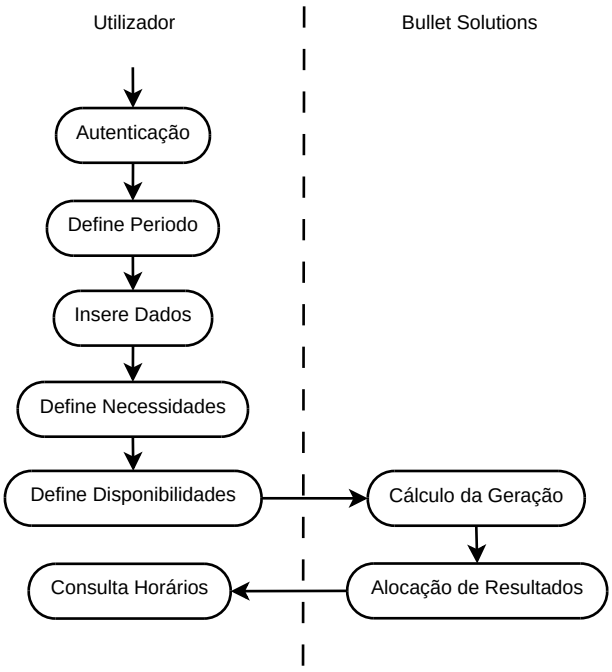


Figura 3.4: Diagrama de Atividade relativo aos casos de uso.

categorias profissionais, colaboradores, turnos e regras. O estado seguinte é onde são definidas necessidades diárias de acordo com os dados inseridos anteriormente. Isto significa que vamos poder definir para um local, um dia e num turno diversas posições. Estas posições por sua vez contêm requisições de qualificações que poderão ser obrigatórias ou preferenciais ou então a requisição de um colaborador específico. Antes de lançar a geração para obter uma solução, ainda é definida a disponibilidade semanal para cada colaborador. Depois de lançado o pedido de geração, é corrido o algoritmo e após encontrada a solução é enviada uma notificação ao utilizador para este poder consultar o horário.

3.2 Bases de Dados

A Base de Dados é uma componente de extrema importância neste projeto, uma vez que é nela que toda a informação é inserida e alterada e é a ela que se acede para a recolha de toda a informação que o algoritmo precisa para elaborar uma solução. Antes da sua descrição, é importante referir que o seu desenvolvimento teve sempre em vista uma estrutura de fácil adaptação e alteração de qualquer componente desejada no futuro.

Além disso, com vista a evitar faltas de compatibilidade com outros produtos de possíveis clientes, o projeto estará preparado para facilmente se adaptar a essas diferenças. Por

exemplo, apesar da estrutura principal ser desenvolvida em SQL Server, facilmente se adaptará a MySQL ou Oracle devido à arquitetura das três camadas. Uma vez que a camada de dados é separada das restantes camadas do projeto, se surgir um cliente com um gestor de base de dados que não seja SQL Server, apenas teremos de implementar a base de dados e as instruções para o novo sistema. O acesso à BD, bem como toda as configurações de alteração de sistema, mantêm-se. Por último, o projeto está também preparado para multi linguagem.

O desenvolvimento da estrutura da Base de Dados (BD) iniciou-se com a criação da BPSWeb que faz a gestão dos pedidos e clientes do projeto. Possui cinco principais tabelas: Clientes (*Clients*), Tarefas (*Jobs*), Servidores de Base de Dados (*DatabaseServers*), Servidores de Cálculo (*Workers*) e Utilizadores (*Users*). Os registos são respetivamente relativos aos dados do cliente, às diferentes tarefas (pedidos que o cliente faz), à gestão dos servidores de base de dados, à disponibilidade e estado dos *Workers* e às credenciais dos utilizadores.

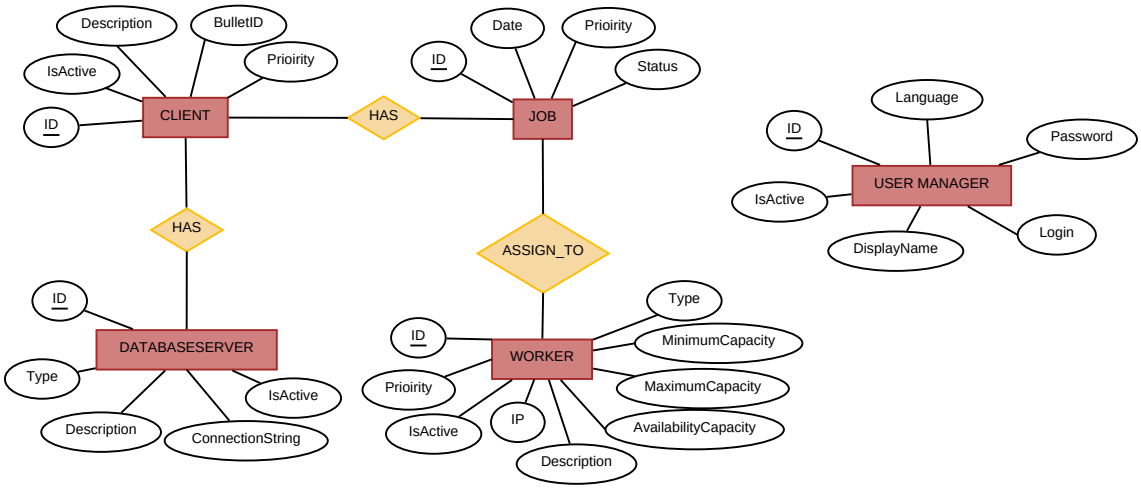


Figura 3.5: Diagrama ER da Base de Dados BPSWeb

A figura 3.5 mostra o Diagrama Entidade-Relacionamento (ER) para a Base de Dados BPSWeb. O diagrama complementa a explicação da importância desta base de dados.

A razão porque a tabela *UserManager* não tem ligação às restantes é porque esta tabela guarda apenas os registos relativos às autenticações da parte da empresa. Todas as outras tabelas se relacionam com os clientes e os pedidos feitos pelos mesmos.

Podemos assim verificar que esta base de dados complementa as arquiteturas anteriormente explicadas, responsabilizando-se pelas ações efetuadas pelos elementos das

mesmas para que o projeto não falhe. Por outras palavras, é nesta base de dados que são guardadas as preferências de cada cliente, tais como qual o servidor de base de dados que utiliza ou qual o identificador do cliente para a Bullet Solutions; os detalhes de cada tarefa que foi solicitada, bem como a que cliente pertence, qual a sua urgência em ser concluída ou qual o *Worker* que a vai atender; quais os servidores de base de dados disponíveis e qual o seu tipo; dados de cada servidor *Worker*: qual o seu tipo, a sua capacidade máxima e a sua capacidade ocupada. Desta forma podemos controlar melhor a situação de cada servidor, bem como o estado dos pedidos de cada cliente, podendo mais facilmente, em caso de um erro, identifica-lo e resolver mais rapidamente o problema.

Como podemos perceber, esta base de dados só por si não faz o projeto funcionar; só faz sentido quando complementada com uma segunda base de dados de cada cliente. Esta segunda base de dados, a que vamos chamar BDClient, contém todos os registos associados a um único cliente e que vão ser fundamentais para o cálculo do escalonamento dos horários. Sempre que a Bullet Solutions receber um cliente novo, o gestor de base de dados cria uma nova BDClient. Esta BD existe por cliente por duas razões fundamentais: a primeira para não ficar sobrecarregada com a quantidade de registos que terá; a segunda para garantir segurança e confidencialidade dos dados (uma vez que será mais fácil garantir desta forma que um cliente não vê ou manipula dados que não lhes pertence).

Vamos agora explicar a sua constituição. Devido à complexidade do projeto e para melhor perceber a sua constituição, o modelo ER foi dividido em vários submodelos.

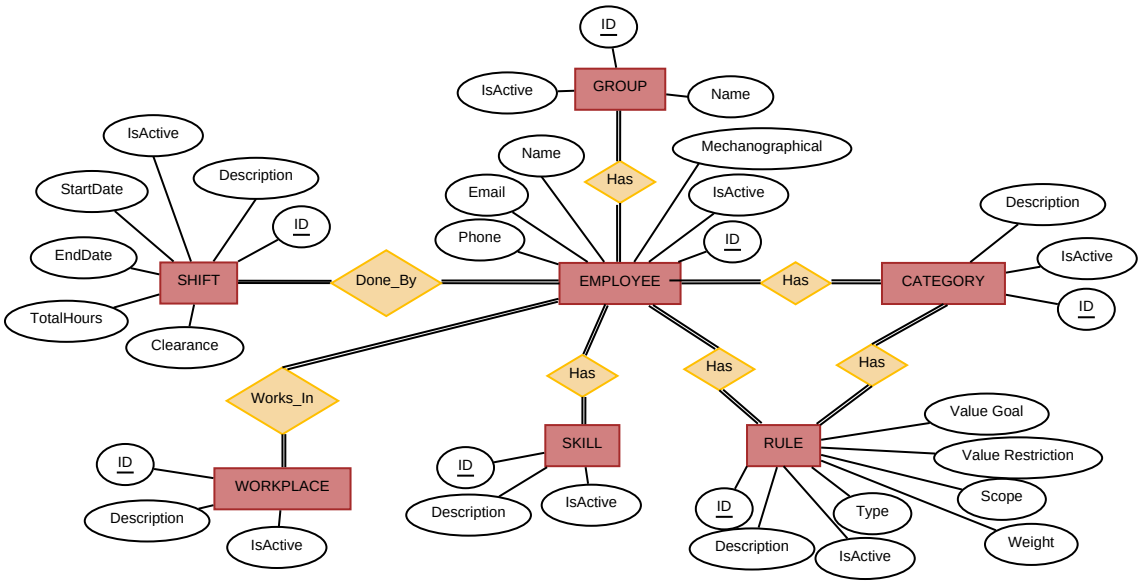


Figura 3.6: Parte do Modelo ER da Base de Dados de um cliente (Configurações Iniciais).

O primeiro, na figura 3.6 a que vamos chamar Modelo das Configurações Inicias, é relativo às tabelas mais simples que correspondem às configurações iniciais inseridas pelo utilizador. São elas: Regras (*Rule*), Locais de Trabalho (*WorkPlace*), Qualificações (*Skill*), Turnos (*Shift*), Categorias Profissionais (*Category*), Colaboradores (*Employee*) e Grupos de Colaboradores (*Group*).

Depois existem tabelas mais específicas e complexas (ver figura 3.7): Ausências dos Colaboradores (*Absence*), Matriz de Custo Turno/Categoria Profissional (*Cost Matrix*), Banco de Horas (*Time Bank*), Disponibilidade dos Colaboradores (*Availability*) e Necessidades (*Need*).

As duas últimas são as que vão ter mais influência na geração da solução. As disponibilidades porque ditam em que horas e em que dias da semana o colaborador pode ou não trabalhar. As necessidades porque definem restrições e objetivos: para um local de trabalho, num determinado dia e turno, escolhemos o recurso a um colaborador específico ou a determinadas qualificações pretendidas. De notar que as tabelas que neste diagrama não têm atributos são as que já foram definidas no diagrama anterior.

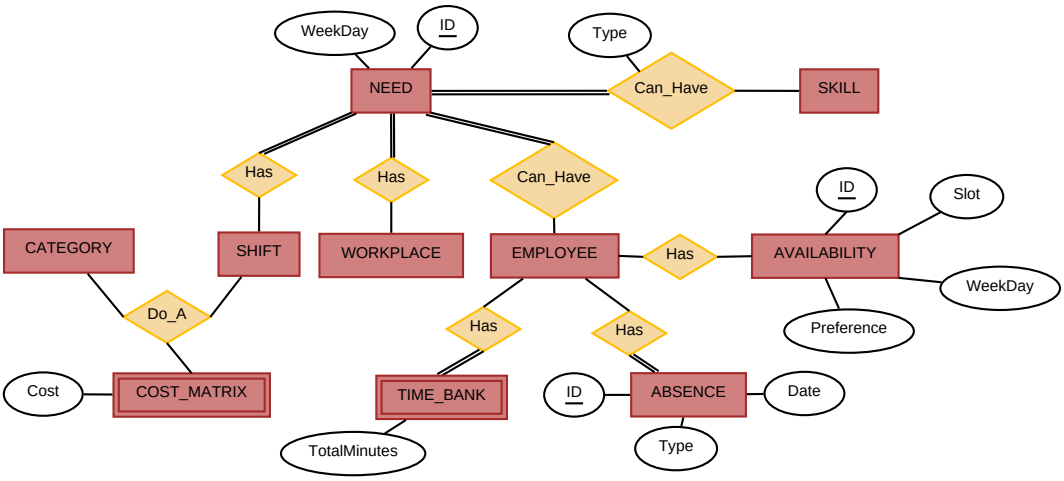


Figura 3.7: Parte do Modelo ER da Base de Dados de um cliente (Configurações Secundárias)

Um outro submodelo são as tabelas cujos registos não são adicionados pelo utilizador, mas sim pelo algoritmo para depois o *FrontEnd* ler e mostrar (figura 3.8): Alocações (*Allocation*, que corresponde à agenda com a solução final) e gerações (*Generation*, estado em que se encontra o cálculo das alocações).

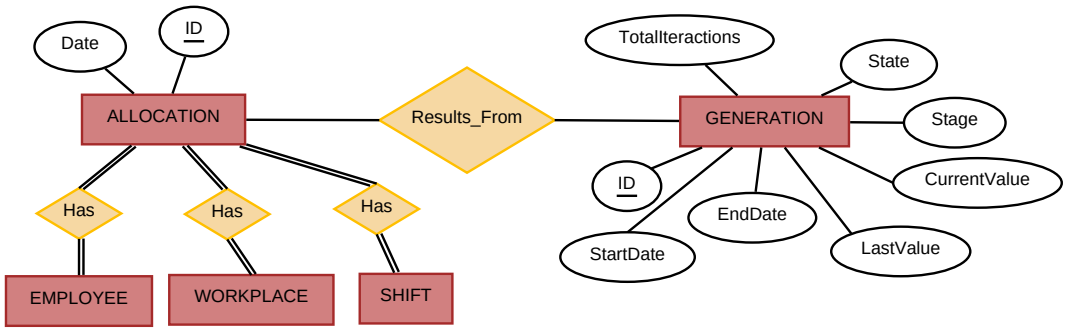


Figura 3.8: Parte do Modelo ER da Base de Dados de um cliente (tabelas geradas pelo algoritmo).

É de realçar que todas as tabelas descritas até aqui estão relacionadas com a tabela Período, como mostra a figura 3.9. Esta tabela cujas colunas são ID, Name, StartDate, EndDate e IsClosed definem o intervalo de tempo para o qual é pretendido gerar um horário. Isto é importante porque qualquer solução que se pretenda vai depender sempre do intervalo de tempo que é definido. Se pensarmos, por exemplo, na geração de um horário escolar, este varia sempre de semestre para semestre, e para cada um deles todos os registos (como as salas, os professores, as disciplinas ou as disponibilidades) podem ser alterados.

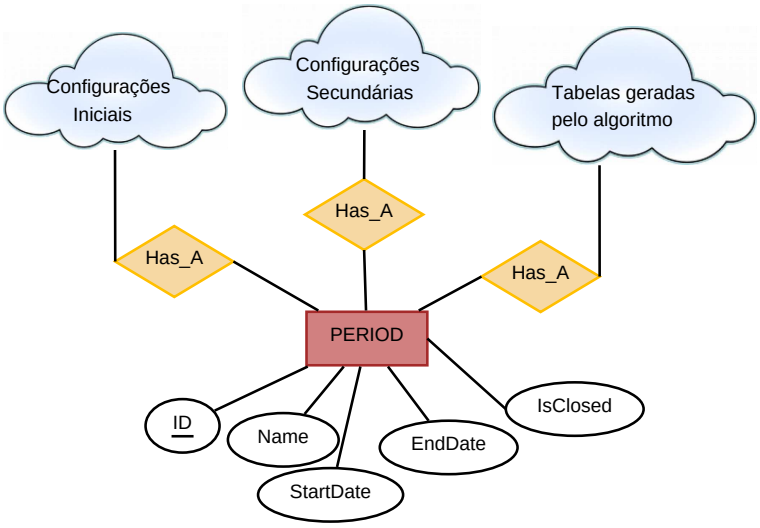


Figura 3.9: Relacionamento da tabela Periodo com as restantes tabelas.

Existe ainda outro grupo de tabelas para fazer toda a gestão dos utilizadores do cliente específico (figura 3.10). São as tabelas que guardam as credenciais de cada utilizador (tabela Utilizadores, *User*), as suas permissões (Perfis, *Profile*), as páginas a que o perfil

tem acesso (Páginas, *Page*) e a quem são enviadas determinadas notificações (tabela Destinatários das Notificações, *Notification Recipient*). Esta última tabela não está relacionada com nenhuma das outras tabelas para tornar possível notificar um conjunto de contactos que não tenham qualquer relação na utilização do sistema. Uma vez que estas tabelas são independentes do intervalo de tempo, nenhuma delas está relacionada com a tabela Período.

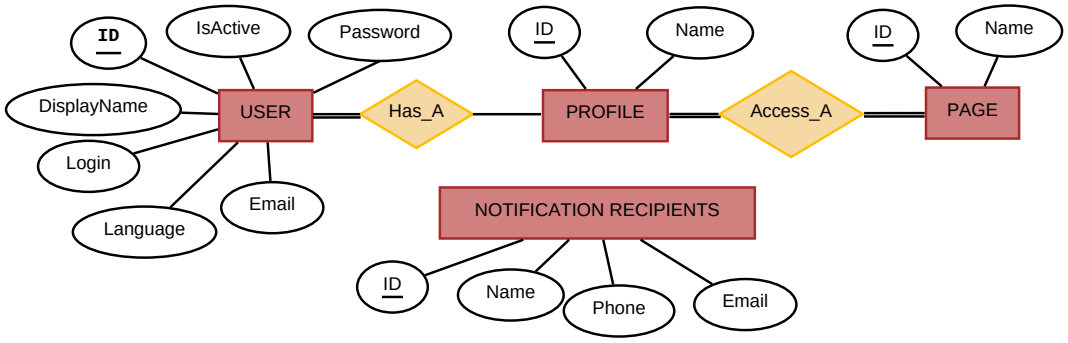


Figura 3.10: Parte do Modelo ER da Base de Dados de um cliente.

Tal como acontece com a BD BPSWeb, existem também na BD do Cliente um conjunto de tabelas para preparar a interface para multi linguagem. Chamadas de tabelas *Locale*, são tabelas que fazem a tradução dos registos previamente guardados na tabela e que não são inseridos pelo utilizador. Um exemplo é o caso da tabela referente aos dias da semana. A língua escolhida por omissão foi a língua inglesa. Assim, a tabela *WeekDays* contém duas colunas, *ID* e *Description* que naturalmente são os dias da semana em inglês. A tabela *WeekDays_Locale* terá as colunas *ID* (com uma chave estrangeira para fazer corresponder ao ID da primeira tabela), *Language* e *Description*.

A imagem 3.11 mostra o desenho das tabelas, as suas colunas e chaves (do lado esquerdo), os registos da tabela *WeekDays* (a meio) e os registos da tabela *WeekDays_Locale* à direita.

Como a tabela dos Utilizadores tem uma coluna onde é guardada a preferência da língua de cada utilizador, sempre que este se autenticar, é guardada essa preferência em Sessão. Depois, sempre que for necessária a consulta a uma tabela que tenha tradução, é passada a língua respetiva, permitindo assim que a *query* retorne o que é pedido na língua certa.

Caso haja novas línguas, apenas é necessário acrescentar na tabela *WeekDays_Locale* as respetivas descrições.

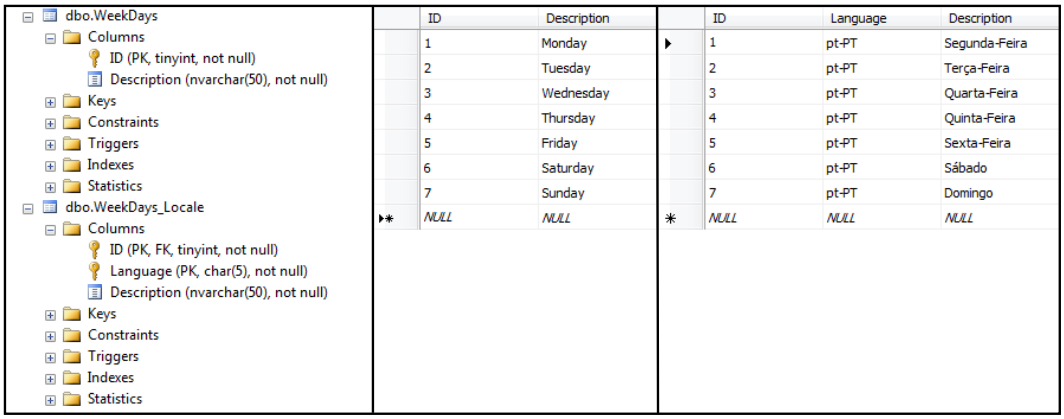


Figura 3.11: Tabelas *WeekDay* e *WeekDay_Locale* e respetivos registos

Ambas as BDs fazem uso de diversas tecnologias essenciais para acelerar qualquer processo de acesso à mesma: Indexação, Restrições e Procedimentos Armazenados.

Indexação (*Indexing*) é uma estrutura de dados que tem como objetivo acelerar e, consequentemente, otimizar as consultas às base de dados [16]. Podemos compreender melhor este conceito se pensarmos no índice de um livro, em que consultamos pelo tema para mais facilmente encontrarmos a página. Da mesma forma, em base de dados, um índice aponta para uma coluna específica de uma tabela para que, aquando da pesquisa de determinados registos, seja mais fácil e rápido encontrar o que se pretende.

Um caso prático onde foi muito usada a indexação foi nas tabelas que têm a coluna *IsActive*. Por exemplo, do ponto de vista de manipulação de Colaboradores para efeitos de alocação num Local de Trabalho, só nos interessa resgatar aqueles que estejam “ativos”. Para isso, na *query* à BD, será forçado o uso de um índice previamente criado com essa mesma coluna, fazendo desta forma com que, em vez de serem percorridos todos os registos pelo ID do Colaborador e depois verificada a coluna IsActive, sejam percorridos os registos que contém esta coluna com o valor *True*.

Restrições (*Constaints*) são regras que se criam para as ações de registos na BD. Isto é, uma tentativa de manipulação de registos que viole uma restrição é abortada [17].

Exemplos de uso de *Constraints* foi para obrigar o preenchimento de determinados campos que podem ser nulos apenas em algumas situações. É o caso da tabela dos Turnos onde só é permitido que um registo seja inserido ou editado se as colunas *StartBreakTime* e *EndBreakTime* forem ambas nulas ou ambas não nulas, de forma a que uma pausa do turno não seja eterna.

Procedimentos Armazenados (SP, *Stored Procedures*) são sub-rotinas em sistemas de base de dados relacionais que executam comandos SQL. Com SP's, em vez das *queries* à BD serem chamadas no código, são guardadas no gestor de base de dados (neste caso, SQL Server) e chamadas posteriormente. As vantagens da sua utilização relacionam-se com a verificação de eventuais erros de sintaxe que se possam cometer e também com a otimização da *query*, através de ferramentas como o Plano de Execução [18].

O Plano de execução foi uma ferramenta valiosa na medida em que sempre que uma SP foi criada, este plano foi usado para mostrar os gastos de tempo com os índices usados, podendo, desta forma, mais facilmente fazer otimizações. Os Procedimentos Armazenados foram das quatro, a tecnologia mais utilizada. Existem dezenas de SP's pois qualquer acesso seja de consulta, edição, inserção ou remoção de dados é feita através dela.

Capítulo 4

Implementação

Neste capítulo vamos descrever com algum detalhe todo o processo de implementação do projeto. Para isso, começamos com uma secção explicativa sobre as tecnologias usadas e posteriormente descrevemos o desenvolvimento das funcionalidades presentes, bem como as técnicas utilizadas para esse efeito.

4.1 Tecnologia

As ferramentas e tecnologias utilizadas no desenvolvimento do projeto foram as seguintes:

- **Microsoft Visual Studio** [19], um IDE desenvolvido pela Microsoft. Ao longo dos anos e com as crescentes atualizações e versões tornou-se numa ferramenta muito completa, uma vez que permite desenvolver diversas aplicações, como por exemplo, Windows Forms (Aplicações para ambiente Windows), Aplicações Web, aplicações para Windows Phone e Serviços Web. Permite também programar em várias linguagens, como Visual Basic, C, C# e C++. A escolha desta ferramenta dependeu não só da empresa (uma vez que é politica trabalhar com o VS), mas também do facto de permite integrar diversos serviços, ter um ambiente gráfico muito intuitivo e ser de fácil desenvolvimento. A versão usada foi a de 2010 uma vez que é versão mais atualizada cuja empresa possui licença.
- **SQL Server** [20], um sistema de gestão de base de dados relacional desenvolvido pela Microsoft, tem como principal objetivo armazenar dados e possibilitar a sua consulta e alteração a partir de aplicações. Desenvolvido em C++, este gestor de base de dados possui diversas versões. A utilizada neste projeto (mais uma vez, por questões de licença) foi a 2008 R2. Esta ferramenta inclui ainda um vasto conjunto de opções que fornecem serviços para auxiliar a gestão da base de dados, por

exemplo o Visual Studio que inclui suporte nativo para programação com SQL Server, tornando muito vantajoso a nível de facilidade de configurações o uso destas duas ferramentas em conjunto.

- **C#** (C-Sharp) [21], uma linguagem de programação declarativa, funcional e orientada a objetos, foi desenvolvida pela Microsoft e aprovada pela ECMA e ISO. Tem-se tornado numa linguagem muito popular por programadores *web*, uma vez que é bastante simples e orientada a objetos. Fornece suporte aos princípios da engenharia de Software por ser de verificação de tipagem forte, deteção de uso de variáveis e recolha automática de lixo. Tem também suporte para internacionalização.

- **Model View Controller** (MVC) [22], um dos modelos de desenvolvimento do ASP.NET (plataforma de desenvolvimento de aplicações web) para desenvolvimento de páginas *web*. A sua filosofia assenta, como o próprio nome indica, em três pilares: Modelo, Vista e Controlador. O Modelo representa o núcleo da aplicação, gere o comportamento dos dados do domínio desta, responde a pedidos de aplicação sobre o seu estado (normalmente feitos pela Vista) e a instruções de alteração do estado (geralmente feitos pelo Controlador); a Vista é responsável da gestão da informação que é mostrada; o Controlador interpreta as ações do rato ou do teclado executadas pelo utilizador e para cada caso informa o Modelo e/ou a Vista.

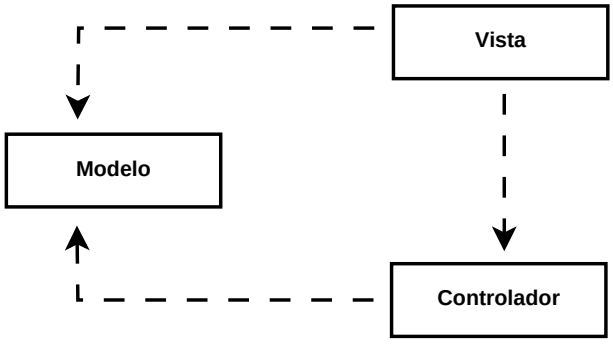


Figura 4.1: Estrutura do Modelo MVC

Através da análise da figura 4.1, que representa a estrutura deste modelo, podemos verificar que a Vista e o Controlador dependem do Modelo, no entanto este não depende dos dois primeiros. Esta independência permite que o modelo seja construido e testado independentemente da apresentação visual, verificando-se um grande benefício na medida em que existe uma separação entre a lógica da interface e a lógica do negócio, que vai de encontro à arquitetura descrita anteriormente.

A partir desta tecnologia, surge-nos outro conceito chamado **Razor** [23]. Trata-se de uma sintaxe de programação ASP.NET usada para criar páginas web dinâmicas, também lançada pela Microsoft Visual Studio 2010, que permite que ao construir uma página HTML seja embebido código C# ou Visual Basic. O código, baseado em servidor, pode ser criado em tempo real enquanto a página é escrita para o *browser*. Quando uma página *web* é chamada, o servidor executa esse código dentro da página antes de retornar a mesma para o *browser*. Se correr pelo servidor, o código é capaz de executar tarefas complexas, como acesso à BD.

- **JQuery** [24], uma biblioteca JavaScript desenvolvida para simplificar os *scripts* do lado do cliente de uma página HTML. Entre as inúmeras funcionalidades destacam-se os efeitos de animação, a manipulação de eventos, a resolução de incompatibilidade entre navegadores e a simplicidade do código. É por isto muito útil no desenvolvimento de uma página *web* intuitiva e simplificada.

4.2 Desenvolvimento

Antes da explicação do começo do desenvolvimento, relembramos que o BPSWeb contém três principais componentes: *Worker*, *Scheduler* e *FrontEnd*. Existe também um módulo *Core*, ao nível da camada de dados, que contém implementações comuns às três componentes e por isso está ligado a todos esses. É o caso da definição de constantes para as traduções, das funções de acesso à base de dados e das funções auxiliares a estas: *Records* e *Mappings*. Ambas são funções que mapeiam os registos que são trazidos da BD, no entanto, as primeiras guardam esses registos em memória, enquanto as segundas não, o que as torna mais rápidas. *Mappings* são definidos como interfaces e usam a função *Reader* como suporte, enquanto *Records* são definições de classes úteis quando queremos manter os registos, libertando os recursos associados à base de dados. Em suma, *Records* foram usados nas funções cujos dados precisavam de ser guardados para mostrar ao utilizador; todas as restantes consultas foram auxiliadas por *Mappings* devido à sua maior eficiência.

O projeto *Worker* corresponde ao desenvolvimento do algoritmo. Como já foi referido, *Workers* são servidores que servem para realizar os cálculos que forem precisos, por isso, este projeto é chamado quando é necessário realizar cálculos.

O projeto *Scheduler* complementa a gestão da base de dados BPSWeb. Todos os processos de gestão de clientes, verificação do servidor de BD a utilizar, atribuição de pedidos aos *Workers* e envio de notificações (como emails e mensagens SMS) são da responsabilidade deste componente.

Por fim, o projeto *FrontEnd* é onde são desenvolvidas todas as interfaces gráficas do utilizador. É o projeto que corresponde à camada UI e com o qual o cliente vai lidar diretamente.

Apesar do projeto *Scheduler* ser mais redirecionado para a administração e o *FrontEnd* para os clientes, o seu desenvolvimento ao nível de interface gráfica e conexão à BD foi muito semelhante (através da tecnologia MVC), uma vez que a filosofia de tratamento de dados é a mesma.

As interfaces gráficas são uma importante questão a ter em conta desde o início do desenvolvimento. O termo ambiente *user friendly* de uma interface relaciona-se com usabilidade, que, por sua vez, é a facilidade que qualquer utilizador tem, por mais inexperiente que seja, em utilizar uma aplicação, um programa ou um sítio *web*. O utilizador precisa de se sentir confortável e intuitivamente conseguir lidar com todo o programa, por isso a interface deve possibilitar ao utilizador, rapidamente e sem problemas, obter o que deseja [25].

Com base nestes aspetos, passamos a descrever algumas boas práticas que foram adotadas. Quando nos referimos a uma aplicação *web*, é necessário que haja consistência entre as diferentes páginas, opções, ações, tipo e tamanho da letra, cores, menu, entre outros. Ou seja, se o menu de opções for definido para estar no cabeçalho, deve manter-se assim durante todas as páginas, caso contrário, irá confundir o utilizador. Também é necessário que haja a perceção de quando uma alteração é feita. Com frequência, um utilizador carrega várias vezes num botão, acabando por repetir diversas ações sem saber, uma vez que a aplicação não o informa sobre a ação que provocou. Outro aspeto que tivemos em consideração foi manter *links*, botões e outros atalhos num tamanho visível o suficiente e esconder as características do sistema que possam confundir a atividade do utilizador.

Nos próximos subcapítulos explicamos como implementámos a base de dados, os formulários da componente *Scheduler* e os formulários da componente *FrontEnd* respetivamente.

4.2.1 Base de Dados

A implementação da Base de Dados foi realizada através do gestor SQL Server, começando por uma base de dados definida inicialmente, que cobria substancialmente as necessidades e efetuando ajustes de acordo com a evolução da aplicação e necessidades.

Consequentemente, as tabelas básicas consideradas indispensáveis (como os Servidores de Base de Dados, os Clientes e os *Jobs*, no caso do projeto *Scheduler*, e os Locais de Trabalho, Colaboradores, Qualificações, Categorias e Turnos, no caso do *FrontEnd*) foram

as primeiras a ser criadas. As tabelas mais complexas que contêm chaves estrangeiras tiveram de ser pensadas e discutidas de forma a determinar a solução mais eficiente e eficaz.

As boas práticas adotadas na construção das base de dados são as seguintes:

1. As colunas ID (identificador e chave primária) de cada tabela, sempre que não forem visíveis para o utilizador, têm o tipo de dados GUID (designado *uniqueidentifier* no caso do SQL Server) [26]. Um GUID é armazenado como um valor de 128 bits e normalmente apresentado em 32 carateres hexadecimais. É gerado aleatoriamente, mas devido à sua grande dimensão é altamente improvável que o mesmo número seja gerado duas vezes. Apesar de ocupar mais espaço comparativamente com um tipo de dados inteiro, decidimos usar GUIDs por permitir mais facilmente a união de registos de diferentes base de dados. Ou seja, podemos criar diferentes registos de diferentes formas, porque ao juntar, a probabilidade de haver colisões de identificadores iguais é muito reduzida.
2. As colunas com registos numéricos devem ter o comprimento mínimo possível, de forma a poupar espaço na memória.
3. As colunas de Descrição e outros textos têm o tipo de dados *nvarchar*, uma vez que este é capaz de armazenar dados *Unicode* (importantes devido à diversidade de carateres na língua portuguesa) e também por permitir uma grande quantidade de carateres [27].

A figura 4.2 representa algumas tabelas criadas, nomeadamente a tabela das Qualificações (*Skills*) onde podemos observar que a coluna ID tem o tipo de dados *uniqueidentifier* e a Descrição o tipo *nvarchar* com comprimento máximo de 50. Esta tabela tem uma chave estrangeira para a tabela dos Períodos, onde podemos verificar a mesma prática no tipo de dados.

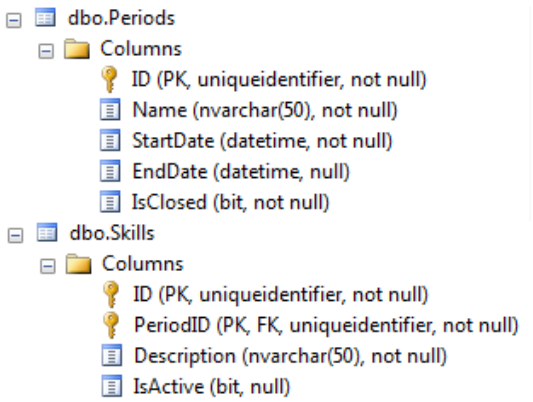


Figura 4.2: Exemplo de duas tabelas criadas no SQL Server

Com o decorrer do projeto, a base de dados foi evoluindo de acordo com as necessidades que foram surgindo, tanto ao nível das tabelas e suas relações, bem como ao nível das SP's. A par do desenvolvimento dos diferentes formulários, foram desenvolvidas as *queries* para o que se pretende, sempre recorrendo à Indexação e ao Plano de Execução para otimizar ao máximo o acesso à BD.

Na figura 4.3 podemos observar duas instruções SQL diferentes. Ambas retornam o ID das Alocações e o ID e a Descrição dos Turnos para uma data específica. No entanto, na instrução do lado esquerdo é usada a chave primária (`PK_Allocations`) e na instrução do lado direito é usado o índice previamente criado que contém a coluna *Date* (`IX_Allocations_Date`), uma vez que é a que vai ser usada para fazer a seleção dos registos.

```
SELECT
    Allocations.ID,
    Allocations.ShiftID,
    Shifts.Description
FROM
    Allocations with (nolock, index(PK_Allocations))
INNER JOIN
    Shifts
    on Allocations.ShiftID = Shifts.ID
WHERE
    Date = '2014-01-13'
```

```
SELECT
    Allocations.ID,
    Allocations.ShiftID,
    Shifts.Description
FROM
    Allocations with (nolock, index(IX_Allocations_Date))
INNER JOIN
    Shifts
    on Allocations.ShiftID = Shifts.ID
WHERE
    Date = '2014-01-13'
```

Figura 4.3: Instrução SQL com uso da chave primária (lado esquerdo) e do índice (lado direito).

Para compreendermos as diferenças que há entre as duas instruções, foram testadas as duas e analisado o Plano de Execução, visível na figura 4.4. Podemos verificar a importância do uso dos índices na obtenção de melhores resultados a nível do custo de CPU bem como o tamanho das linhas estimado. Se o número de registos fosse maior, melhores resultados a este nível poderíamos observar.
Verificamos ainda o uso do *NOLOCK* [28]; o SQL Server utiliza o *LOCK* como um mecanismo de garantia da integridade dos dados, isto é, por omissão o SQL Server bloqueia o conteúdo de uma tabela quando existem múltiplos acessos à mesma informação. Por exemplo, se quisermos consultar os registos de uma tabela que ao mesmo tempo esteja a ser atualizada, a tabela vai ficar bloqueada. A solução é analisar se queremos realmente consultar os dados que podem ou não estar atualizados e, em caso afirmativo, usar o *NOLOCK*.

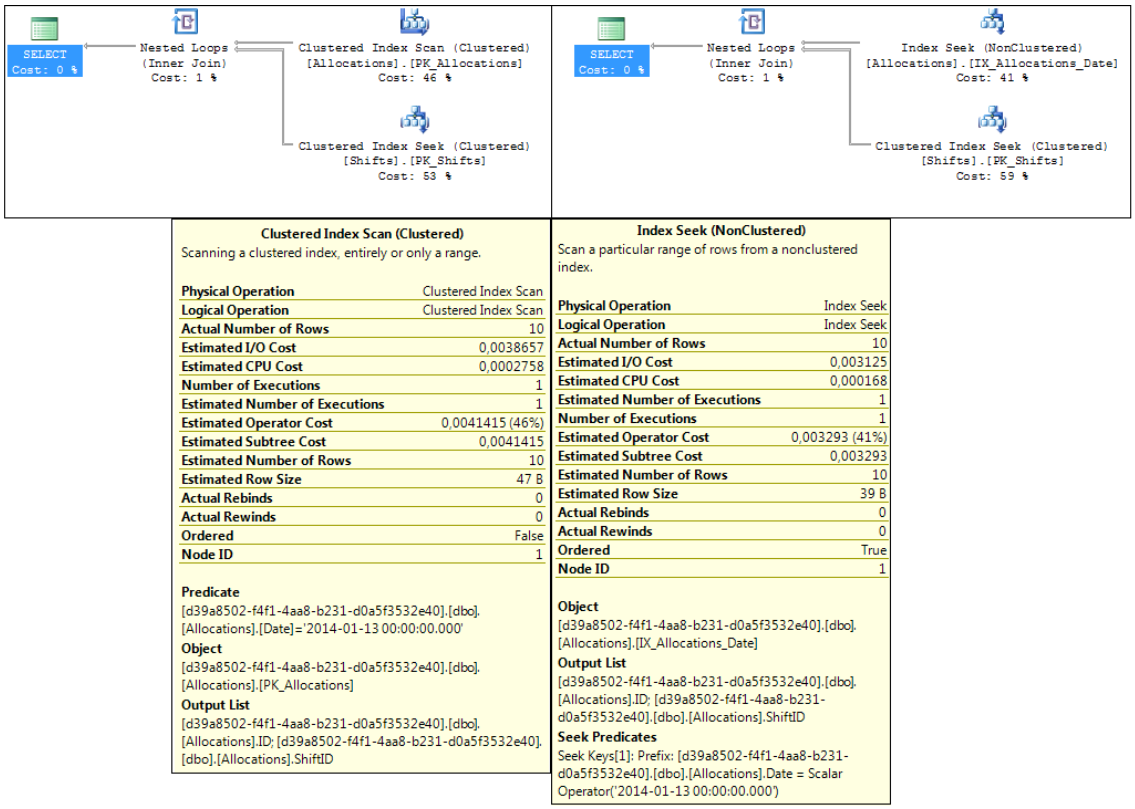


Figura 4.4: Plano de execução para a instrução com uso da chave primária (lado esquerdo) e do índice (lado direito).

4.2.2 Formulários *Scheduler*

Como já foi referido, a componente *Scheduler* faz a gestão interna do projeto e define como se comportará o projeto face aos diversos pedidos de diversos clientes. Para isso, é definida uma classe chamada *BusinessLogic* com as funções que são executadas sempre que um cliente faz um pedido. Mais especificamente, vamos supor um caso em que um Cliente faz o pedido de geração do horário pretendido. O esquema na figura 4.5 é a explicação do comportamento da classe *BusinessLogic* desde que um cliente executa um pedido até que o servidor o processa.

Um administrador do projeto muitas vezes deseja executar determinadas ações, tais como: verificar o estado de cada cliente, saber qual o servidor de base de dados que cada cliente utiliza, conhecer os servidores de base de dados disponíveis, quais os *Workers* disponíveis e suas caraterísticas (prioridade, capacidade de processamento dos pedidos e tipo), quais os pedidos de cada cliente ou fazer a gestão de utilizadores.

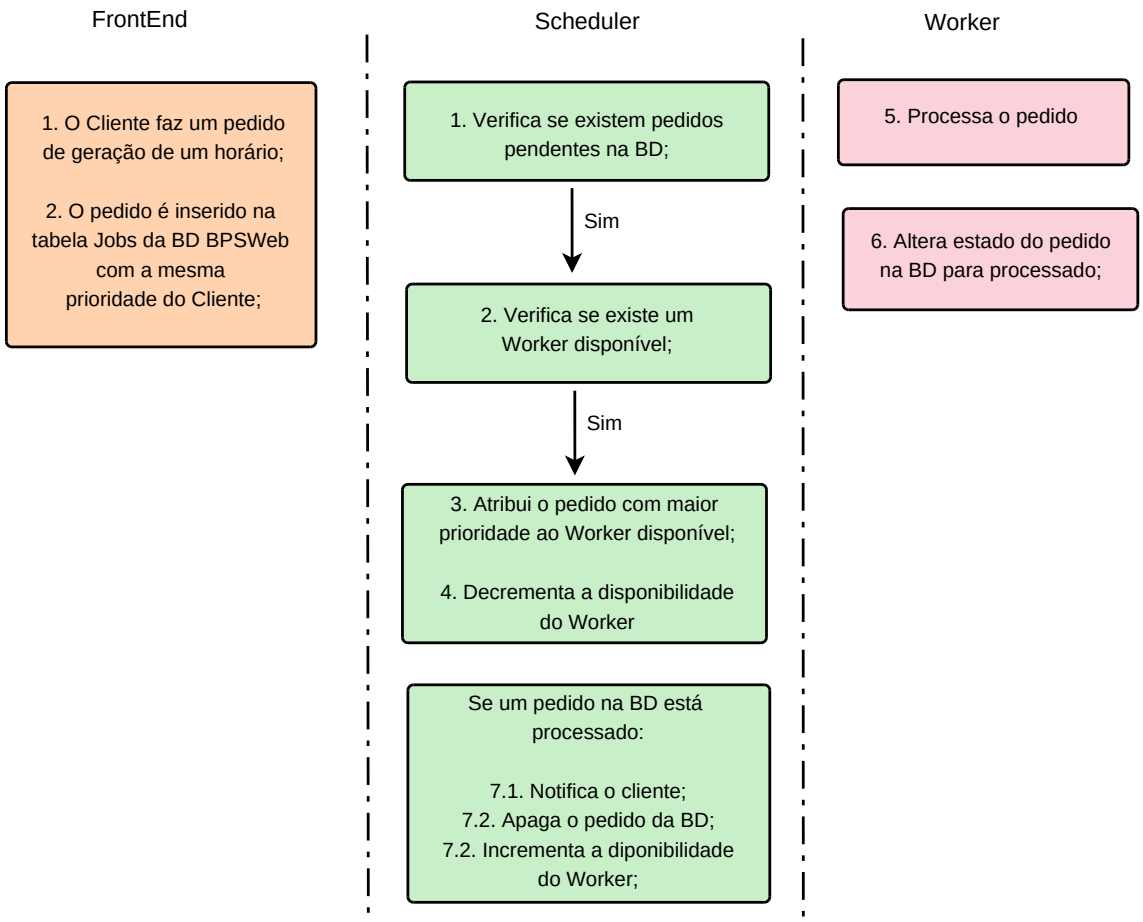


Figura 4.5: Comportamento da classe *BusinessLogic* no *Scheduler*

Como não é prático estar constantemente a alterar o código ou a correr manualmente as instruções SQL, para esse efeito foi criada uma interface gráfica de auxílio aos administradores.

Para isso, foram criados formulários para cada componente a consultar: Clientes, *Workers*, Servidores de Base de Dados e Utilizadores. Cada um desses formulários respeita a lógica MVC: no Modelo criámos as componentes correspondentes às colunas de cada tabela; no Controlador criámos as funções que pretendemos para cada componente; na Vista desenvolvemos como queremos mostrar os resultados.

Vamos perceber melhor esta lógica através do exemplo do formulário dos Clientes.
O Modelo deste formulário é uma classe com a definição de cada coluna na respetiva BD: ID, Descrição, Ativo, Prioridade e ID Bullet. Contém ainda a descrição do Servidor de Base de Dados pois para a visualização na Vista é importante que o utilizador veja a descrição e não o ID.

O segundo passo foi definir no Controlador as funções do que pretendemos: consultar todos os clientes, inserir um novo cliente, editar um cliente existente, consultar os detalhes do cliente e remover um cliente. Por uma questão de organização de código as funções contidas no Controlador chamam funções que estão numa classe denominada repositório e que existe também por cada formulário.

Quando, nas funções nos repositórios, queremos fazer chamadas à base de dados, essa função chama uma outra contida no ficheiro que agrega todas as funções relativas à base de dados. Cada uma delas, por questões de segurança, não contém a instrução SQL, mas sim o nome do Procedimento Armazenado na base de dados. Especificamente, no exemplo de consultar os detalhes de um cliente, vamos ter uma função no controlador, que chama uma função no repositório que por sua vez vai mapear o resultado de uma função de consulta SQL. Essa consulta retorna a Descrição, o estado de Ativo, a Prioridade e o ID Bullet de um cliente, recebendo para isso a chave primária (ID).

No terceiro passo vamos mostrar o resultado da execução desta função. Neste momento, os dados definidos no Modelo estão preenchidos, uma vez que foi feito o mapeamento do resultado da instrução SQL para os dados do Modelo. Desta forma, na Vista foram criados objetos HTML para fazer a leitura dos dados no modelo e mostrar na interface o seu conteúdo, concluindo assim o resultado dos detalhes do cliente.

Com o objetivo de tornar o projeto mais compacto e fácil de manter, foram criadas *interfaces*. Uma interface contém definições para um conjunto de funcionalidades relacionadas que uma classe ou estrutura podem implementar. Através de interfaces, torna-se possível incluir determinados comportamentos de diferentes fontes numa classe. Isto é bastante importante na linguagem C# uma vez que esta não suporta herança múltipla de classes [29]. Sendo assim, sempre que é chamada uma função na classe repositório ou no ficheiro SQL, na verdade vamos chamar a sua interface que contém a definição das funções implementadas na classe principal.

4.2.3 Formulários *FrontEnd*

A lógica de implementação dos formulários de listagem, consulta dos detalhes, inserção, edição e remoção de registos segue a mesma lógica descrita na subsecção anterior, assim como o recurso a interfaces. Assim, para cada um dos formulários, existe um modelo, uma vista e um controlador.

O bloco de código na Tabela 4.1 representa as funções no Repositório para quando queremos editar um Local de Trabalho. Temos uma função que chama a função respetiva no ficheiro de base de dados (*GetWorkPlaceByIDPeriodID(IDatabaseClient Database, Guid ID, Guid PeriodID)*) e outra que mapeia os resultados nos registos que queremos (*MapWorkPlace(GetWorkPlaceRecord DBWorkPlace, Guid PeriodID)*).

```
public WorkPlace GetWorkPlaceByIDPeriodID(IDatabaseClient Database, Guid
    ID, Guid PeriodID)
        {
            return MapWorkPlace(Database.GetWorkPlaceByIDPeriodID(ID,
                PeriodID), PeriodID);
        }

private WorkPlace MapWorkPlace(GetWorkPlaceRecord DBWorkPlace, Guid
    PeriodID)
        {
            WorkPlace WorkPlace = null;

            if (DBWorkPlace != null)
            {
                WorkPlace = new WorkPlace
                {
                    ID = DBWorkPlace.ID,
                    PeriodID = PeriodID,
                    Description = DBWorkPlace.Description,
                    IsActive = DBWorkPlace.IsActive
                };
            }
            return WorkPlace;
        }
```

Tabela 4.1: Funções no Repositório para quando o utilizador pretende editar um local de trabalho.

Como resultado e à semelhança do exemplo dos Clientes no *Scheduler* temos a Base de Dados a retornar o ID do Local, o ID do Período, a Descrição e se está Ativo e a guardar essa informação no modelo anteriormente criado.
Por fim, a vista da edição dos Locais de Trabalho (Tabela 4.2) vai conter objetos HTML com a informação atual e os campos possíveis de ser editáveis para o utilizador poder proceder à alteração. Podemos verificar que temos uma legenda “Descrição” na língua respetiva do utilizador e depois um campo editável com o conteúdo atual da Descrição no Modelo.

```
@using (Html.BeginForm())
{
    @Html.HiddenFor(model => model.ID)
    @Html.HiddenFor(model => model.PeriodID)

    <fieldset class="round5">
        <legend class="round3">@LS.Edit @LS.WorkPlace</legend>

        <p>
            <label class="noRoundBot">@LS.Description</label>
            @Html.EditorFor(model => model.Description)
            @Html.ValidationMessageFor(model => model.Description)
        </p>
        <p>
         @if (Model.PeriodID == Constants.DummyPeriodID)
         {
            <label class="noRoundBot">@LS.IsActive</label>
            @Html.EditorFor(model => model.IsActive)
            @Html.ValidationMessageFor(model => model.IsActive)
         }
        </p>
        <p>
            <input type="submit" value = "@LS.Save" />
            @Html.ActionLink(LS.BackToList, "Index", "WorkPlace", new {
                PeriodID = Model.PeriodID }, new { @class = "a-button
                round3" })
        </p>
    </fieldset>
}
```

Tabela 4.2: Excerto do código em Razor HTML para mostrar a edição de um local de trabalho

Resumidamente, o que resultou deste processo foi uma página com o ID do Local e o ID do Período escondidos e com a Descrição e o estado Ativo visíveis de um local de trabalho que o utilizador informou que quer editar. Depois de alterados os dados pretendidos e do utilizador carregar no botão, o processo será idêntico, apenas com a diferença de em vez de um Método HTTP GET, será um método HTTP POST, o que por sua vez implica que em vez de solicitarmos registos, vamos envia-los para serem processados [30].

No caso dos formulários que dependem destes, como o caso dos Colaboradores que têm categorias e regras ou dos Grupos que têm diversos Colaboradores ou ainda das Necessidades que é composto por Locais de Trabalho, Turnos, Colaboradores e Qualificações, foram precisos outros cuidados. Nas instruções SQL foi necessário recorrer a JOINS (junção de informação de uma ou mais tabelas recorrendo a um valor comum

a essas tabelas) e UNION (operação para unir o resultado de duas instruções SQL diferentes). Em alguns casos também foi preciso recorrer a outras tecnologias como jQuery, o que é explicado seguidamente.

4.2.4 Formulários JQuery

O formulário das Disponibilidades tem uma particularidade, consiste numa tabela com os dias da semana na primeira linha e as horas na primeira coluna. O objetivo é sabermos para cada colaborador, quais os dias da semana e horas que está disponível para trabalhar, podendo ter quatro hipóteses: indisponível, a evitar, não preferencial e preferencial, pintando para isso as células respetivamente das cores vermelho, cor-de-laranja, amarelo e verde, como mostra a figura 4.6.

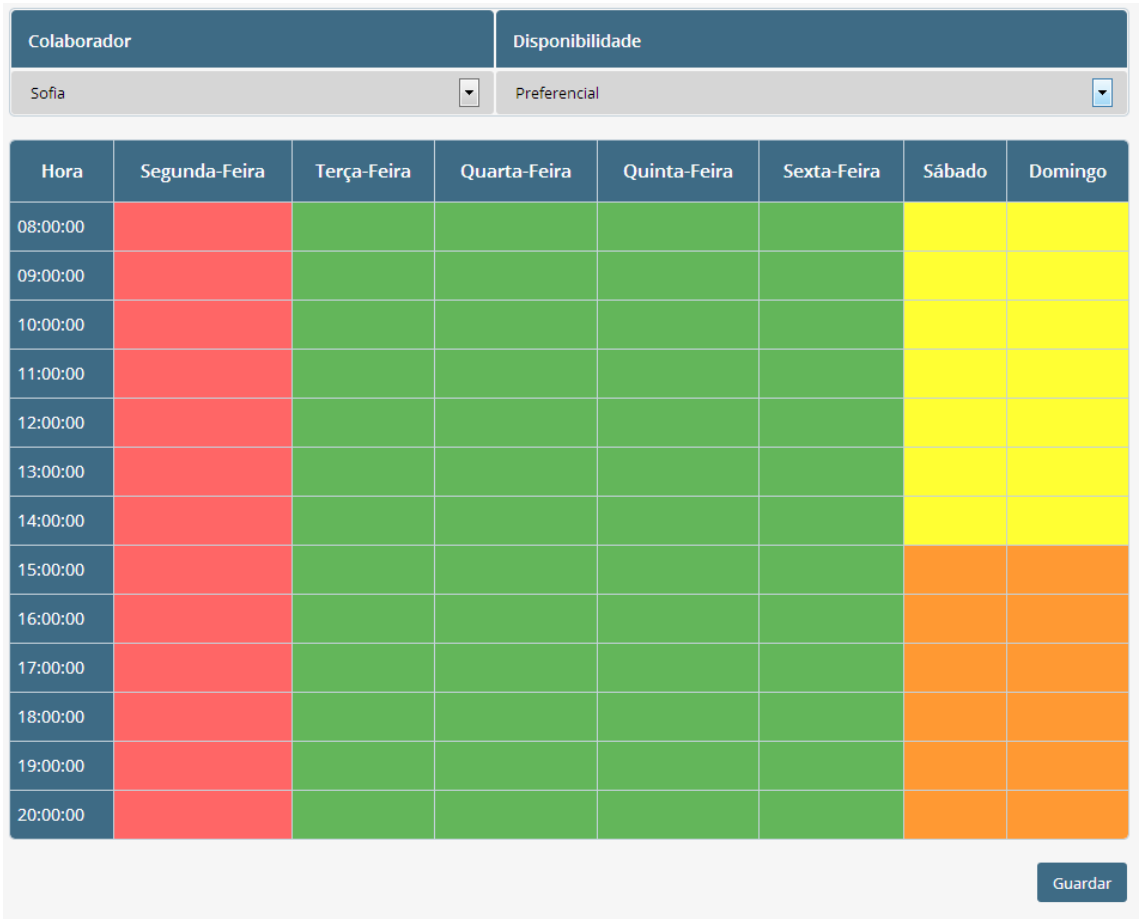


Figura 4.6: Interface gráfica do formulário das Disponibilidades

De notar que na imagem, o intervalo entre cada tempo é de exatamente uma hora, mas poderia ser mais específico e saber as disponibilidades de 15 em 15 minutos e, nesse

caso, o número de células ia quadruplicar. Para que isto seja possível e de forma a que o desempenho do formulário seja o melhor, foram criadas duas funções em JQuery: A primeira é executada quando se clica na *dropdownlist* com a escolha da disponibilidade e guarda o código RGB previamente associado à opção; a segunda é para quando clicamos numa célula qualquer. Por omissão as células vêm todas pintadas de verde, pois apenas são guardados na tabela da BD os registos que sejam de cor diferente da verde. Quando a célula for clicada, o primeiro passo é ir buscar a posição da célula, verificando para isso o número da linha e o número da coluna. Depois, a posição é inserida numa *string* para posteriormente o Controlador associar a posição ao dia da semana e à hora e dessa forma guardar o registo na base de dados, juntamente com a cor guardada no primeiro passo.

JQuery foi usado para efeitos semelhantes, como o aparecimento de calendários aquando da inserção de uma data.

4.2.5 Preparação para multi linguagem

Como foi referido no capítulo das base de dados, existem tabelas desenvolvidas para auxiliar nas traduções. Também no projeto existe uma classe criada para o mesmo efeito e que conjuga com essas tabelas. As classes são muito simples e consistem apenas em declarar as *Strings* da palavra/frase que irá aparecer na interface do utilizador. Cada *string* tem um identificador (ID) do tipo *inteiro* e que deverá corresponder ao mesmo ID na base de dados. Assim, sempre que for necessário aparecer determinada palavra, é feita uma pesquisa à base de dados pelo ID da *String* e pelo Idioma do utilizador (relembramos que é guardado em Sessão logo aquando da sua autenticação). Como exemplo, vamos supor que temos a seguinte frase que queremos que apareça em português: ”Palavra-Passe errada!”. Na base de dados do cliente, na tabela *LocalizeStrings* está guardada a frase com o *ID = 23 e Description = ”Wrong Password!”* e na tabela *LocalizeStrings_Locale* estará: *ID = 23, Language = ”pt-PT”e Description = ”Palavra-Passe errada!”*. Na Vista onde queremos que apareça esta frase, apenas teremos de chamar a *String* cujo ID seja 23.

Capítulo 5

Resultados

Nesta secção estão descritos os resultados finais do projeto *Scheduler* e do *FrontEnd*.

5.1 Formulários da componente *Scheduler*

A figura 5.1 representa a interface gráfica que serve de auxílio ao controlo do projeto e que só é acedida pelos programadores e administradores do mesmo, não havendo qualquer acesso para um utilizador cliente. Podemos observar que é possível configurar quais os clientes que têm acesso ao BPSWeb, bem como quais os servidores de base de dados e os *Workers* disponíveis e ainda quais os utilizadores do sistema.

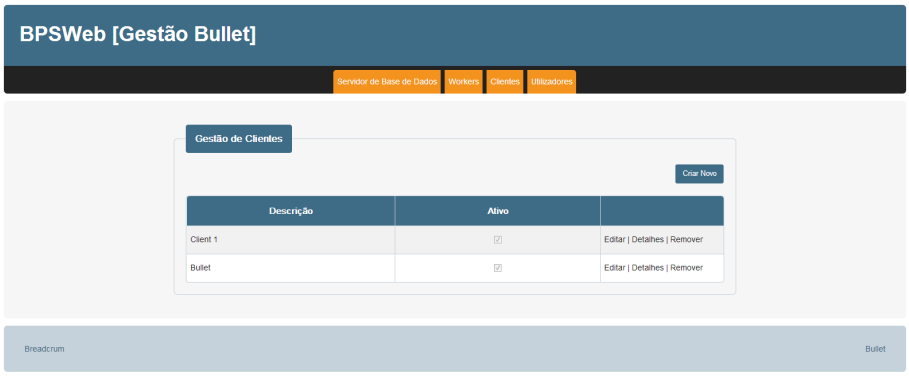


Figura 5.1: Interface gráfica do projeto *Scheduler*.

5.2 Formulários básicos da componente *FrontEnd*

Passando para o projeto com o qual o cliente final irá lidar, temos na figura 5.2 o formulário de inserção de um novo período, indicando para isso a data início e a data fim do mesmo. Este deve ser o primeiro passo que o utilizador faz quando pretende gerar um novo horário.

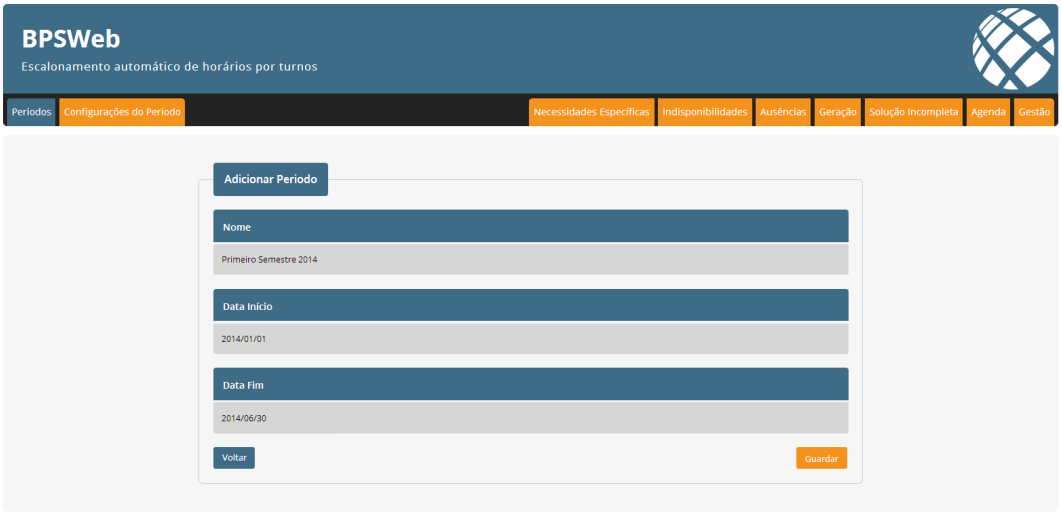


Figura 5.2: Interface gráfica do formulário de inserção de um novo período.

Na figura 5.3 podemos observar o formulário de listagem dos locais de trabalho. A partir daqui as imagens que serão mostradas são apenas do corpo da página em questão, uma vez que o cabeçalho e rodapé com o menu são exatamente iguais. De notar que este e outros formulários que dependam do período estão no mesmo menu por uma questão de coerência da interface gráfica.

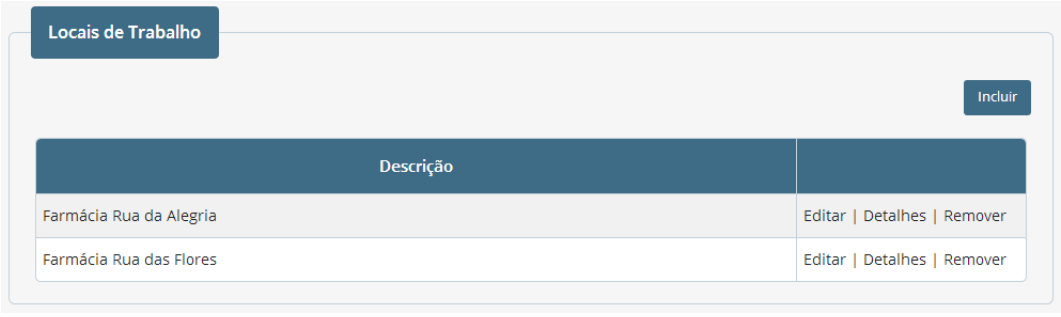


Figura 5.3: Interface gráfica da listagem dos Locais de Trabalho

5.3 Formulários secundários da componente *FrontEnd*

O formulário da matriz de custo é representado na figura 5.4. Este não é um formulário que seja contabilizado no cálculo do algoritmo. Podemos considera-lo como um extra para ajudar o utilizador a fazer escolhas de determinado colaborador para um turno, caso o custo do mesmo seja um fator a ter em conta.

Figura 5.4: Interface gráfica da matriz de custo turno/categoria profissional.

Quando queremos definir posições para uma determinada necessidade, temos o formulário das necessidades (figura 5.5).

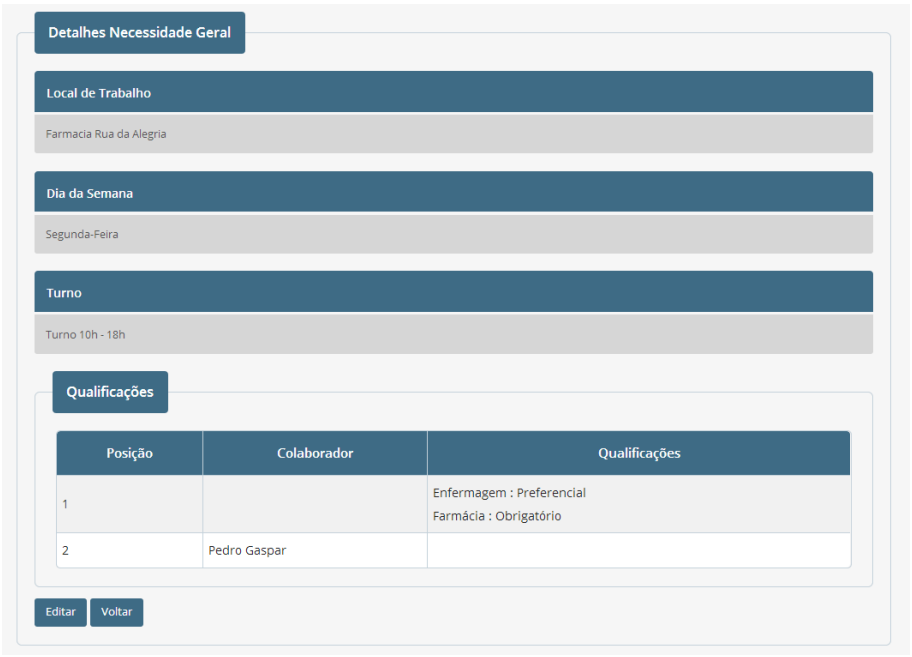


Figura 5.5: Interface gráfica dos detalhes das Necessidades.

Como podemos verificar, na Farmácia da Rua da Alegria, todas as segundas-feiras no turno das 10h às 18h, são precisas duas posições: uma com alguém que obrigatoriamente tenha a qualificação de farmácia e que, se possível, também tenha qualificações de enfermagem; a outra exige o colaborador Pedro Gaspar. De notar que podíamos ter quantas posições desejássemos e outro turno ou local de trabalho que anteriormente tivesse sido definido para o período em questão.

Se uma geração está em curso, isto é, se o *Worker* está a executar o algoritmo em busca de uma solução pretendida, então nenhum formulário de um período específico pode ser alterado e no formulário das gerações podemos verificar o estado da mesma. Na Figura 5.6 podemos verificar o estado da geração: qual a data início e fim da solução final, qual o estado da geração, qual a etapa (Solução inicial ou otimização), entre outras. De notar que a data da última geração é um campo que é automaticamente atualizado frequentemente para nos certificarmos que a geração continua em curso e que não está bloqueada.

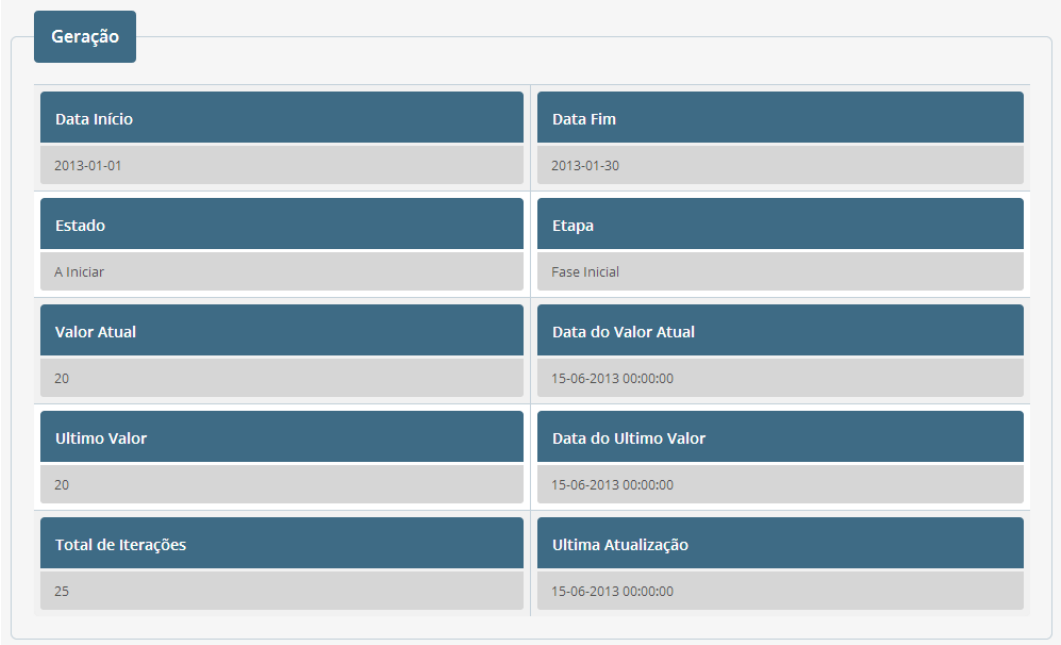
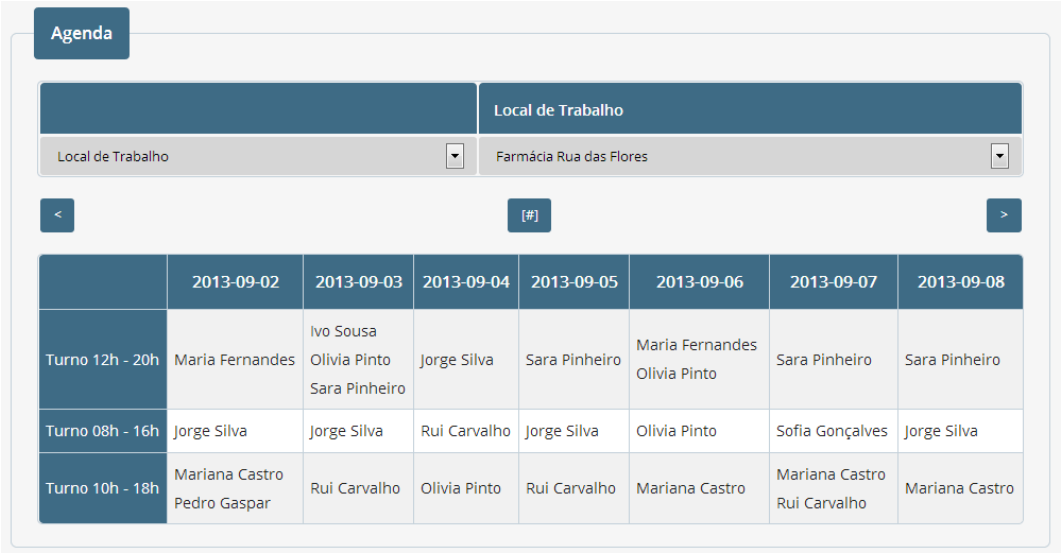


Figura 5.6: Interface gráfica de uma geração em curso.

Por fim, quando a geração termina e é encontrada uma solução, podemos consultar o formulário Agenda onde encontramos o horário filtrado por um colaborador, por um local de trabalho ou por um turno. No exemplo da imagem 5.7 podemos verificar a semana corrente em que nos encontramos; para o Local de trabalho selecionado (Farmácia da Rua das Flores) e para cada um dos turnos ativos nesse intervalo de tempo, verificamos quais são os colaboradores que estão a trabalhar.
Para permitir que o utilizador faça consultas consoante a sua preferência, a agenda



	2013-09-02	2013-09-03	2013-09-04	2013-09-05	2013-09-06	2013-09-07	2013-09-08
Turno 12h - 20h	Maria Fernandes	Ivo Sousa Olivia Pinto Sara Pinheiro	Jorge Silva	Sara Pinheiro	Maria Fernandes Olivia Pinto	Sara Pinheiro	Sara Pinheiro
Turno 08h - 16h	Jorge Silva	Jorge Silva	Rui Carvalho	Jorge Silva	Olivia Pinto	Sofia Gonçalves	Jorge Silva
Turno 10h - 18h	Mariana Castro Pedro Gaspar	Rui Carvalho	Olivia Pinto	Rui Carvalho	Mariana Castro	Mariana Castro Rui Carvalho	Mariana Castro

Figura 5.7: Interface gráfica da Agenda filtrada por um local de trabalho.

além do filtro por local de trabalho também permite os filtros de turno e de colaborador, oferecendo desta forma uma melhor visualização consoante a necessidade.

Poderão existir casos em que não existe uma solução possível de resolver todas as necessidades. Vamos supor um caso em que se define que é necessário um colaborador com uma determinada qualificação e nesse mesmo dia todos os colaboradores qualificados estão indisponíveis. Neste caso, em vez da solução final, o utilizador é redirecionado para o formulário da Solução Incompleta onde poderá tomar uma das seguintes decisões: ou atribui um colaborador que não tem a qualificação pretendida mas que está disponível para trabalhar nesse turno ou então elimina essa necessidade. A figura 5.8 mostra o formulário da Resolução da Solução Incompleta.

Data	Local de Trabalho	Turnos	
2013-09-02	Farmácia Rua das Flores	Turno 10h - 18h	Atribuir Colaborador | Remover
2013-09-02	Farmácia Rua das Flores	Turno 12h - 20h	Atribuir Colaborador | Remover
2013-09-06	Farmácia Rua das Flores	Turno 12h - 20h	Atribuir Colaborador | Remover
2013-09-06	Farmácia Rua das Flores	Turno 08h - 16h	Atribuir Colaborador | Remover
2013-09-07	Farmácia Rua das Flores	Turno 08h - 16h	Atribuir Colaborador | Remover

Figura 5.8: Interface gráfica da Resolução da Solução Incompleta.

5.4 Comparação com *Softwares* Existentes

Antes do começo do desenvolvimento do BPSWeb, e como descrito no capítulo do Estado de Arte, foram estudados e analisados os *Softwares* concorrentes. Verificámos rapidamente que qualquer um deles pecava pela falta de algoritmia no escalonamento de horários. Sendo assim, esta torna-se a principal vantagem do nosso projeto relativamente a esses.
Relativamente à interface gráfica, podemos verificar que o BPSWeb tem uma ambiente intuitivo, uma utilização fluida e com os menus dispostos numa ordem lógica de utilização. O menu horizontal com as principais opções, a mesma base CSS comum a todos os formulários (como os cabeçalhos, os tipos de letra ou a formatação dos botões) tornam-se pontos fortes neste projeto. Funcionalidades como notificações, geração de relatórios e suporte para um grande número de funcionários e turnos não são um problema.

Pontos positivos da concorrência relacionam-se com a importação de ficheiros que, por enquanto, este projeto ainda não dispõe e pelas versões para aplicações móveis que hoje em dia são muito importantes. No entanto, são dois pontos em análise e discussão para trabalho futuro.

Capítulo 6

Conclusão

Ao longo destes sete meses de estágio curricular, foi desenvolvida e concluída uma aplicação *web* capaz de satisfazer as necessidades de um cliente no que diz respeito à sua organização interna.

Os objetivos propostos foram alcançados e foi desenvolvida uma base de dados que sustenta todo o projeto. Obtivemos uma aplicação intuitiva, de fácil utilização para gestão de todos os dados de uma empresa: inserção, remoção, edição e consulta de colaboradores, turnos, regras, qualificações, categorias profissionais, ausências, etc. É possível identificar necessidades e disponibilidades, importantes na geração do algoritmo, bem como consultar o estado da geração e o horário final e ainda resolver soluções incompletas.
O projeto como um todo está preparado para lidar com inúmeros pedidos diferentes de diversos clientes devido à sua implementação a nível de lógica de negócio. A arquitetura implementada permite fáceis alterações a qualquer componente que o cliente queira fazer. Também desenvolvemos uma aplicação de gestão interna para ajudar a administração a fazer um ponto de situação a cada utilização.
Com o BPSWeb atualmente é possível fazer a gestão do pessoal em pequenas e médias empresas e usufruir de um escalonamento automático de horários, estando por isso pronto para receber um cliente teste que ajude este projeto a crescer cada vez mais e melhor.

Quanto ao meu percurso em particular, para alcançar os objetivos, ao longo do estágio fiz um estudo do mercado concorrente, que desenvolve *software* de escalonamento de pessoal automático, de modo a conhecer a área e tirar proveito do que cada um tem de melhor. Aprofundei conhecimentos de programação *web* assim como das tecnologias MVC e SQL Server, apliquei conhecimentos de diferentes conceitos de diversas unidades curriculares e aprendi a utilizar todos esses conceitos num único projeto, sendo mais autónoma, analítica e critica.

Foi muito gratificante todo o percurso ao longo deste estágio curricular. Debati-me muitas vezes com dificuldades ao nível da implementação na camada de negócio e na adaptação das exigências académicas comparativamente às exigências do mercado de trabalho. Aprendi a superar essas dificuldades através de ferramentas de pesquisa e da equipa da Bullet Solutions.

6.1 Trabalho Futuro

Devido à complexidade do projeto ainda há alguns ajustes que se continuam a fazer com o objetivo de tornar o produto mais eficaz. No entanto, devido à preparação do BPSWeb para alterações, os formulários que se encontram em falta, não irão requerer grande tempo de desenvolvimento, estando a finalização do projeto para breve.
Como foi referido, uma vez que este é um produto abrangente a uma grande diversidade de profissões, é possível que sejam efetuadas alterações para obter uma melhor adaptação às circunstâncias de cada cliente.

A segunda versão está já a ser desenvolvida, com novos formulários, como a geração de relatórios e informações e um editor manual para fazer alterações de última hora. Assim que o produto for lançado no mercado é implementada a parte do *Software* com um Serviço. Posteriormente, será pensada a hipótese da aplicação ser adaptada a dispositivos móveis, algo que já está preparado a nível CSS. Testes de usabilidade também fazem parte de um importante trabalho futuro para garantir o ambiente *user friendly* da aplicação.

Capítulo 7

Acrónimos

API	Aplication Programming Interface
BD	Base de Dados
BPSWeb	Bullet Personal Sheduler Web
BTTE	Bullet TimeTabler Education
CPU	Central Processing Unit
CSS	Cascading Style Sheets
ECMA	European Computer Manufacturers Association
ER	Entidade-Relacionamento
GUID	Global Unique Identifier
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
ISO	International Organization for Standardization
MVC	Model View Controller
REST	Representational State Transfer
SaaS	Software as a Service
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
XML	Extensible Markup Language

Referências

[1] BulletSolutions, “A empresa,” http://www.bulletsolutions.com/, Consultado em Agosto de 2013.

[2] D. G. da Administração e do Emprego Público, “Decreto-lei 259/98 sobre os princípios gerais em matéria de duração e horário de trabalho na administração pública,” http://www.dgaep.gov.pt/upload/Legis/1998_dl_259_18_08.pdf, Consultado em Agosto de 2013.

[3] ShiftWorkSolutions, “Employee shift work schedules,” http://shift-work.com/articlesnewsletters/shiftwork-articles/employee-shift-work-schedules-an-introduction/, Consultado em Agosto de 2013.

[4] W. H. O. I. A. F. R. O. CANCER, “Shiftwork,” http://monographs.iarc.fr/ENG/Monographs/vol98/mono98-8A.pdf, Consultado em Agosto de 2013.

[5] Business_Management_Systems, “Snap schedule,” http://www.bmscentral.com/products/schedule/employee-scheduling-software-whatsnew13.aspx, Consultado em Março de 2013.

[6] ShiftPlanning, “Online employee scheduling software,” https://www.shiftplanning.com, Consultado em Março de 2013.

[7] TimeForge, “Label management software and biometric time clocks,” http://www.timeforge.com/site/, Consultado em Março de 2013.

[8] NimbleSchedule, “Online employee scheduling software,” https://www.nimbleschedule.com, Consultado em Março de 2013.

[9] Scheduleit, “Schedule it,” http://www.scheduleit.co.uk/, Consultado em Março de 2013.

[10] I. WhenToWork, “Whentowork,” http://whentowork.com/logins.htm, Consultado em Março de 2013.

[11] TimeCenter, “Timecenter scheduling software,” https://www.timecenter.com/, Consultado em Março de 2013.

[12] SISCOG, “Sistemas cognitivos,” https://www.siscog.pt, Consultado em Março de 2013.

[13] M. Turner, D. Budgen, and P. Brereton, “Turning software into a service,” 2003.

[14] Microsoft, “Uma introdução ao software + serviços, saas e soa,” http://msdn.microsoft.com/pt-br/library/dd875466.aspx, Consultado em Agosto de 2013.

[15] W3C, “Web services architecture,” http://www.w3.org/TR/ws-arch/#service_oriented_architecture, Consultado em Agosto de 2013.

[16] J. W. Jeffrey D. Ullman, Hector Garcia-Molina, “Database Systems: The Complete Book,” 2001.

[17] W3Schools, “Sql constraints,” http://www.w3schools.com/sql/sql_constraints.asp, Consultado em Agosto de 2013.

[18] Microsoft, “Sql stored procedures,” http://msdn.microsoft.com/en-us/library/aa174792(v=sql.80).aspx, Consultado em Agosto de 2013.

[19] ——, “Visual studio,” http://www.microsoft.com/visualstudio/en-gb/products/2010-editions/professional, Consultado em Agosto de 2013.

[20] ——, “Sql server,” http://www.microsoft.com/en-us/sqlserver/product-info.aspx, Consultado em Agosto de 2013.

[21] E. International, “C# language specification,” *June 2006. Retrieved January 26, 2012*, Consultado em Agosto de 2013.

[22] Microsof, “Model view controller,” http://msdn.microsoft.com/en-us/library/ff649643.aspx, Consultado em Agosto de 2013.

[23] S. Guthrie, “Introducing razor – a new view engine for asp.net,” http://weblogs.asp.net/scottgu/archive/2010/07/02/introducing-razor.aspx, Consultado em Agosto de 2013.

[24] jQuery Foundation, “jquery api,” http://api.jquery.com/, Consultado em Agosto de 2013.

[25] Microsoft, “Usability in software design,” http://msdn.microsoft.com/en-us/library/ms997577.aspx, Consultado em Agosto de 2013.

[26] SQLServer, “uniqueidentifier,” http://technet.microsoft.com/en-us/library/ms187942.aspx, Consultado em Agosto de 2013.

[27] ——, “nchar and nvarchar,” http://technet.microsoft.com/en-us/library/ms186939.aspx, Consultado em Agosto de 2013.

[28] ——, “Table hints (nolock),” http://technet.microsoft.com/en-us/library/ms187373.aspx, Consultado em Agosto de 2013.

[29] Microsoft, “Interfaces (c# programming guide),” http://msdn.microsoft.com/en-us/library/vstudio/ms173156.aspx, Consultado em Agosto de 2013.

[30] W3Schools, “Http methods,” http://www.w3schools.com/tags/ref_httpmethods.asp, Consultado em Agosto de 2013.